US006026414A

United States Patent [19]

Anglin

[11] Patent Number: 6,026,414

[45] Date of Patent: Feb. 15, 2000

[54] **SYSTEM INCLUDING A PROXY CLIENT TO BACKUP FILES IN A DISTRIBUTED COMPUTING ENVIRONMENT**

[75] Inventor: **Matthew Joseph Anglin**, Vail, Ariz.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **09/035,526**

[22] Filed: **Mar. 5, 1998**

[51] **Int. Cl.**[7] **G06F 15/173**

[52] **U.S. Cl.** **707/204**; 707/10; 395/200.33; 395/200.47; 395/684; 395/182.11

[58] **Field of Search** 707/9, 10, 204; 395/200.33, 200.36, 200.47, 683, 684, 182.11, 182.09

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,005,122	4/1991	Griffin et al.	709/203
5,408,619	4/1995	Oran	707/10
5,628,005	5/1997	Hurvig	707/8
5,673,381	9/1997	Huai et al.	714/1
5,682,534	10/1997	Kapoor et al.	709/304
5,689,701	11/1997	Ault et al.	707/10
5,745,752	4/1998	Hurvig et al.	707/200
5,845,082	12/1998	Murakami	709/226
5,852,724	12/1998	Glenn, II et al.	709/239
5,857,102	1/1999	McChesney et al.	713/100
5,867,650	2/1999	Osterman	709/203

FOREIGN PATENT DOCUMENTS

0 574 900 A2	12/1993	European Pat. Off. .
0 773 503 A2	5/1997	European Pat. Off. .
2 288 477	10/1995	United Kingdom .

OTHER PUBLICATIONS

ACM Computing Surveys, vol. 22, No. 4, Dec. 1990, "Distributed File Systems: Concepts and Examples" by Eliezer Levy and Abraham Silberschatz.

IBM Corporation's International Technical Support Organization "ADSM Concepts", SG24–4877–00, Feb. 1997, Chapters 1 & 6.

IBM Corporation's ADSTAR Distributed Storage Manager "Using the UNIX Backup–Archive Clients", Version 3, Release 1, SH26–4075–01, Chapters 1 & 3.

IBM Corporation's ADSTAR Distributed Storage Manager for HP–UX "Administrator's Guide", Version 3, GC35–0320–00, Chapters 4 & 12.

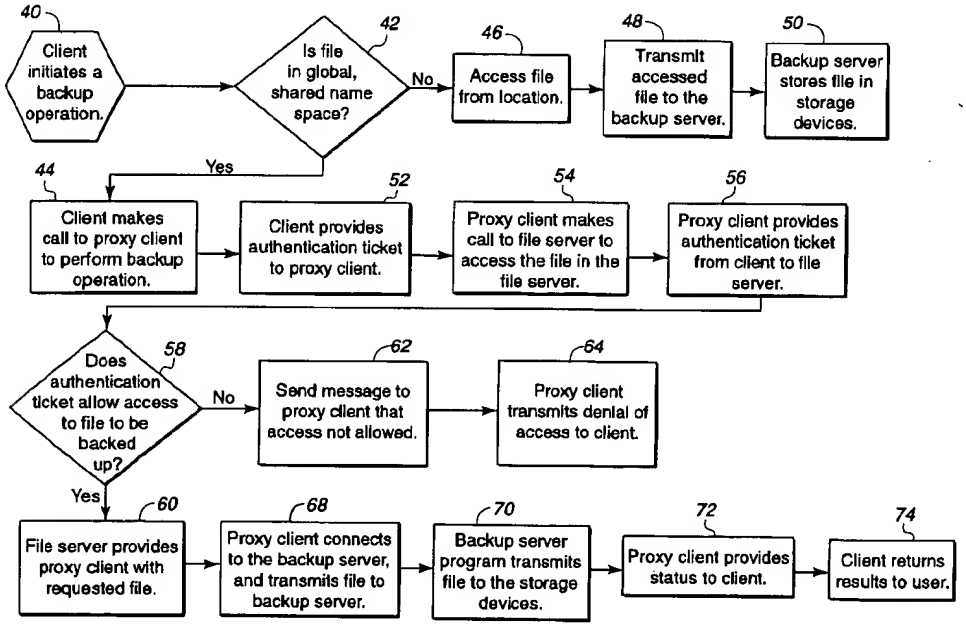
Primary Examiner—Jean R. Homere

Attorney, Agent, or Firm—David W. Victor; Konrad Raynes & Victor

[57] **ABSTRACT**

Disclosed is a system for backing up files in a distributed computing system. A file server maintains files in a shared name space. The file server provides a first backup client program and a second backup client program with access to the files in the shared name space. The first backup client program initiates a backup request to backup a requested file. A determination is made as to whether the requested file is maintained in a shared name space. The backup request is transmitted to the second backup client program upon determining that the requested file is maintained in the shared name space. The second backup client program transmits a message to the file server to provide the requested file. The file server transmits the requested file with the file server to the second backup client program. The second backup client program then transmits the requested file to a backup server program. The backup server program stores the requested file in a storage device.

24 Claims, 3 Drawing Sheets



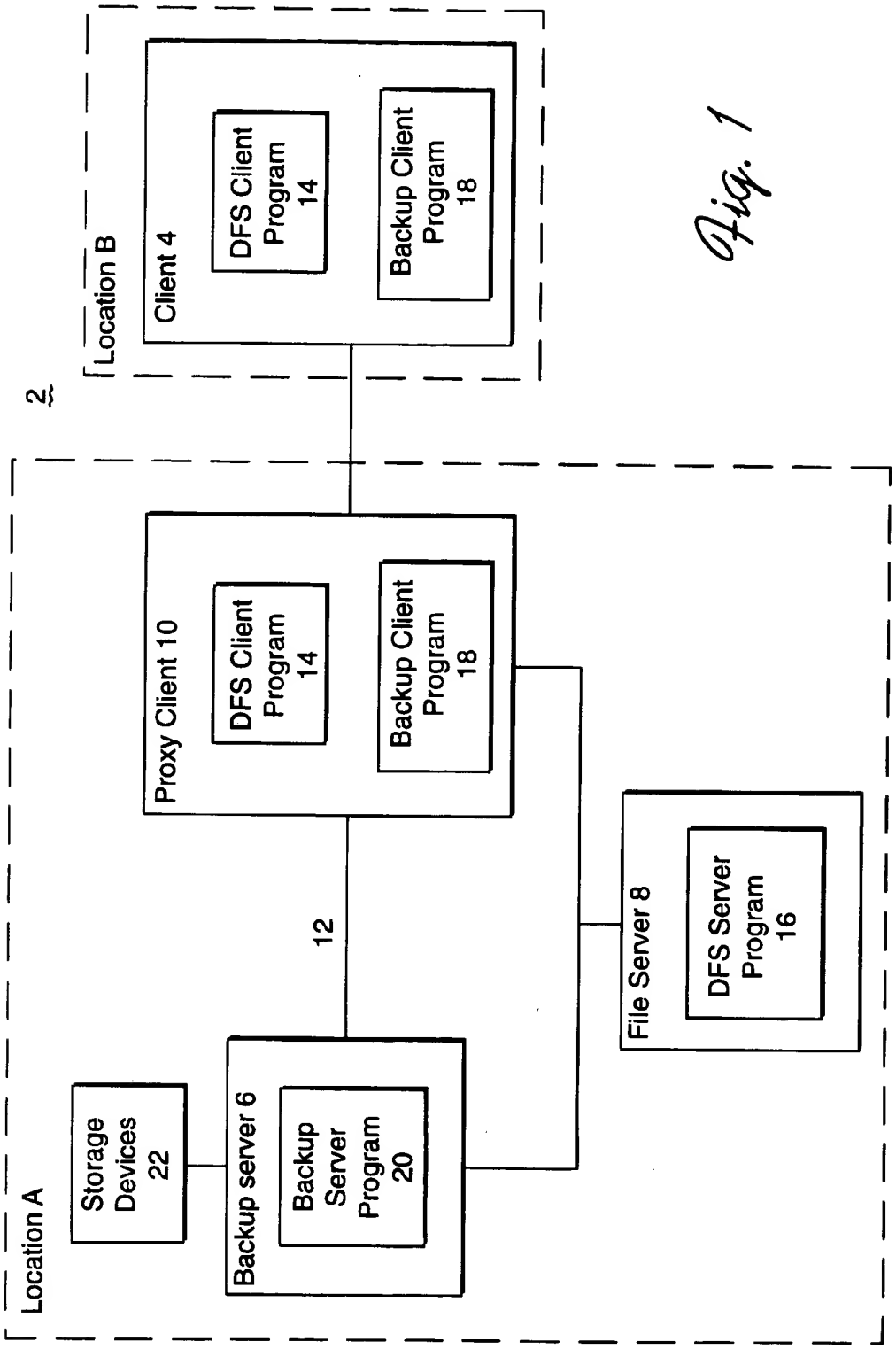

Location A
Location B
Client 4
DFS Client Program 14
Backup Client Program 18
2
Proxy Client 10
DFS Client Program 14
Backup Client Program 18
12
File Server 8
DFS Server Program 16
Backup server 6
Backup Server Program 20
Storage Devices 22

Fig. 1

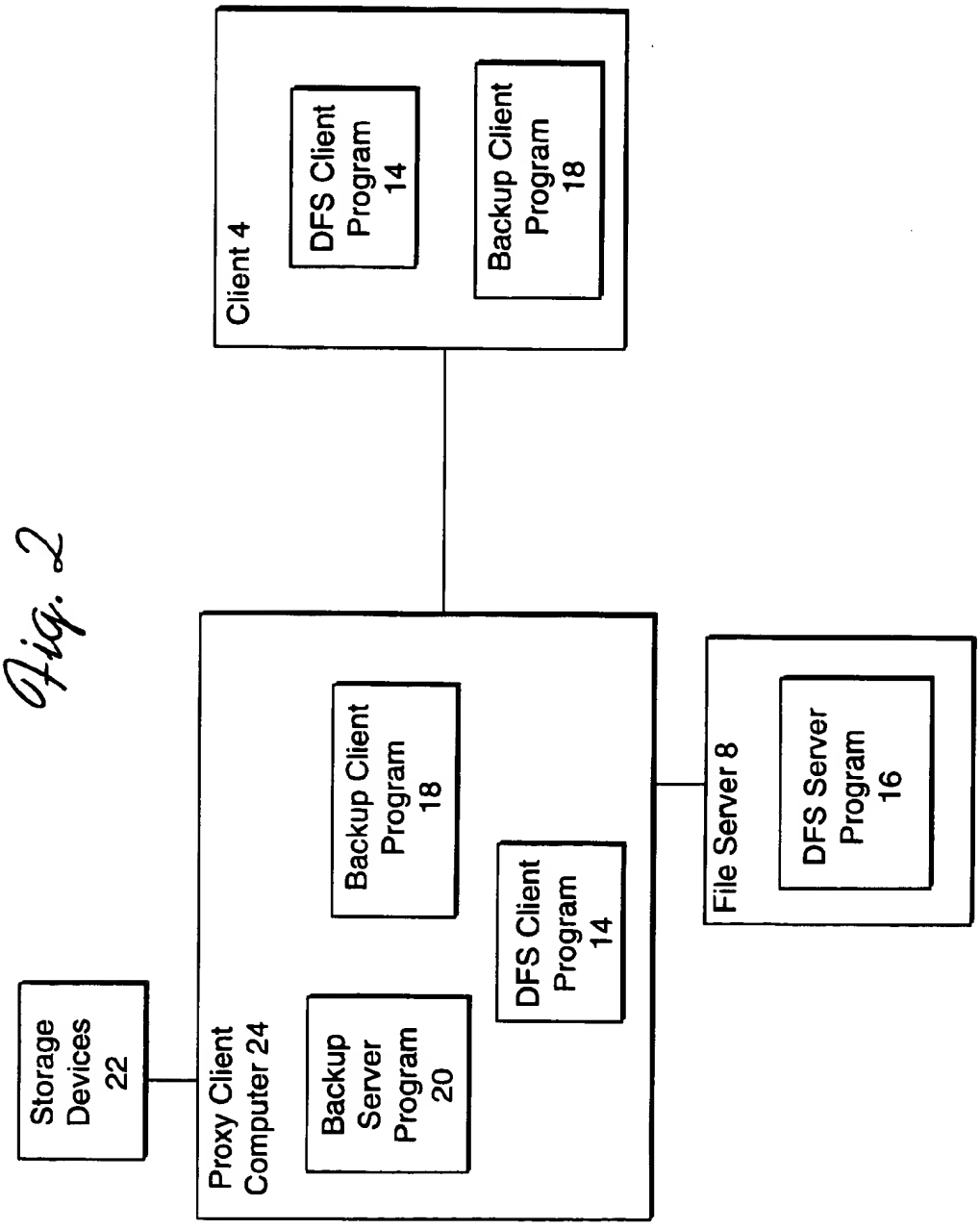

Client 4
DFS Client Program 14
Backup Client Program 18
Proxy Client Computer 24
Backup Client Program 18
DFS Client Program 14
Backup Server Program 20
Storage Devices 22
File Server 8
DFS Server Program 16

Fig. 2

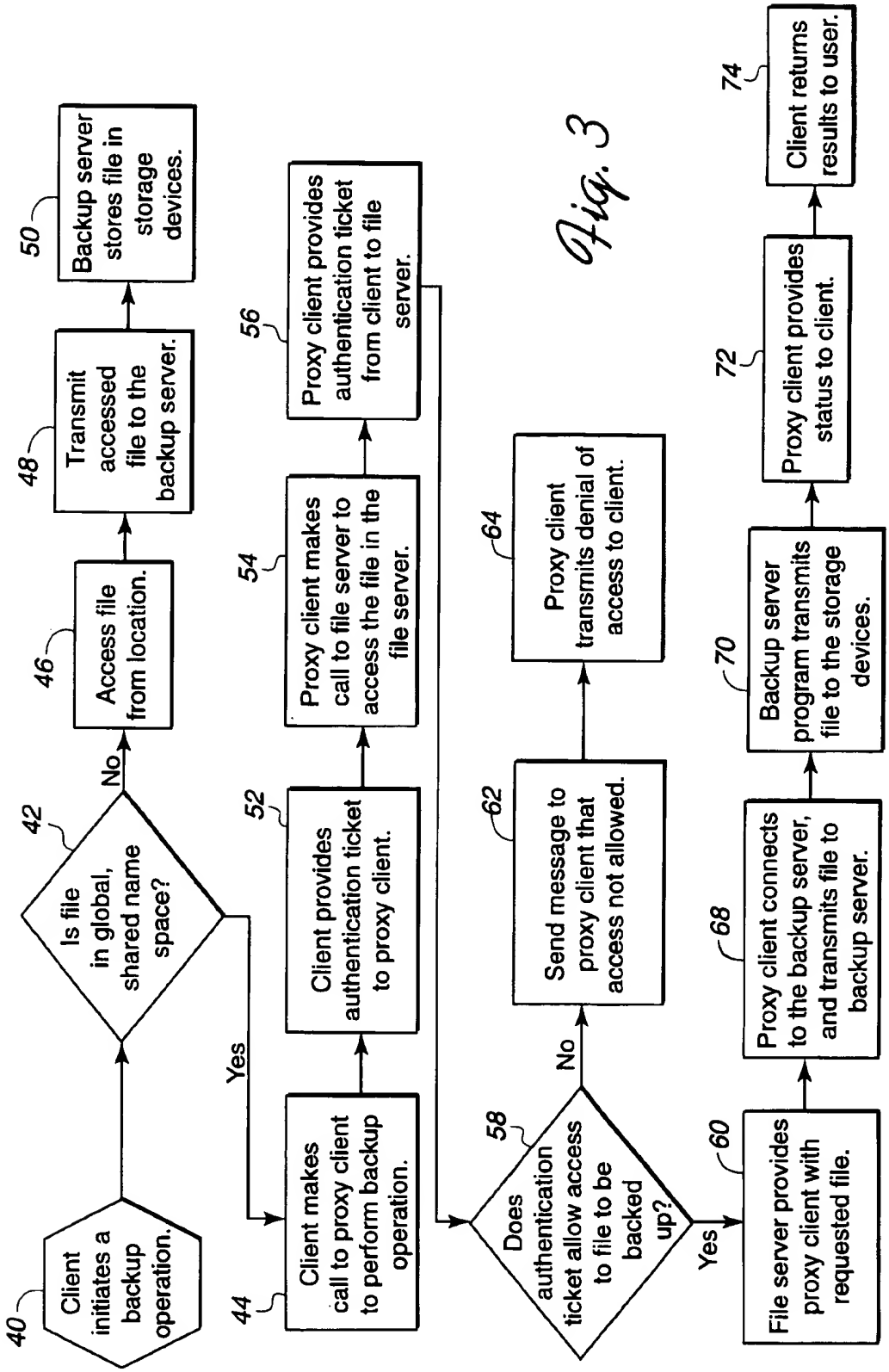

40
Client initiates a backup operation.
42
Is file in global, shared name space?
No
Yes
44
Client makes call to proxy client to perform backup operation.
46
Access file from location.
48
Transmit accessed file to the backup server.
50
Backup server stores file in storage devices.
52
Client provides authentication ticket to proxy client.
54
Proxy client makes call to file server to access the file in the file server.
56
Proxy client provides authentication ticket from client to file server.
58
Does authentication ticket allow access to file to be backed up?
No
Yes
60
File server provides proxy client with requested file.
62
Send message to proxy client that access not allowed.
64
Proxy client transmits denial of access to client.
68
Proxy client connects to the backup server, and transmits file to backup server.
70
Backup server program transmits file to the storage devices.
72
Proxy client provides status to client.
74
Client returns results to user.

Fig. 3

SYSTEM INCLUDING A PROXY CLIENT TO BACKUP FILES IN A DISTRIBUTED COMPUTING ENVIRONMENT

FIELD OF THE INVENTION

Preferred embodiments of the present invention relate to a system for backing up files in a distributed computing system and, in particular, using a proxy client to backup files.

BACKGROUND OF THE RELATED ART

In a distributed computing system, different computers, operating systems, and networks interact as if they were all part of a single system. The file system has a single set of global file names. A particular machine in the system need not know where the file is physically located. Instead, the file may be accessed anywhere in the network using the global file name. Global file names are part of the shared name space which devices within the distributed file system may access. One such distributed file system is the Andrew File System (AFS) available through Transarc, Corporation ("Transarc"). An AFS server performs file mapping between the directory name of a file and the location, making the file space location independent. With file independence, a user at a workstation linked to the network need only know the global file name, which includes the path name, and not the physical location where the file resides.

Another distributed system, is the Distributed File System (DFS), available from Transarc and International Business Machines, Corp. (IBM), which is a component of the Distributed Computing Environment (DCE) standard promulgated by the Open Software Foundation (OSF). IBM is the assignee of the subject patent application. The DFS and AFS systems allow users to access data throughout the network. Any changes made by one user to a file is available to all users. The DFS and AFS systems include security services that provide authentication to limit access to authorized users.

The AFS system offered by Transarc includes a backup program called "butc" (Backup Tape Coordinator). Butc is a volume backup system used to dump volume images to tape devices attached to the file server. However, the minimum backup unit for the butc program is a volume as the butc program does not provide support for file-level backup and recovery.

Hierarchical storage management programs, such as the IBM Adstar Distributed Storage Management (ADSM) product, provide backup/archive support and migrates less frequently used files to backup storage to free space. The ADSM server provides hierarchical storage management backing files up on tape drives, optical disks, and other storage medium. The ADSM backup feature saves copies of files from a client computer to a storage space managed by an ADSM server. Thus, data at a client computer running an ADSM client is protected in the event of data loss due to a hardware or software failure, accidental deletion, and/or logical corruption. With the ADSM program, clients can backup volumes, directories, subdirectories or files. ADSM allows incremental backup of only those files that have been changed. In this way, ADSM avoids the need to do a full dump to backup as only those modified files are backed up. This incremental backup reduces network utilization and traffic. The IBM ADSM product is described in "ADSM Version 2 Presentation Guide," (IBM Document SG24-4532-00, International Business Machines, copyright 1995), which publication is incorporated herein by reference in its entirety.

IBM has combined the ADSM product with AFS and DFS file servers to provide backup support for these products. An AFS or DFS server would include an ADSM client to transfer files to an ADSM server, which then backs up the files in a storage device managed by the ADSM server. One problem with using such backup software in a distributed file system is that the client managing backup operations, such as the ADSM client, must read a file to be backed-up. This reading operation consumes network resources. The ADSM client must then consume network resources again by transferring the file it has read from the file server to the ADSM server. Network traffic is further increased if the ADSM client is on a separate machine from the AFS/DFS server. The IBM publications entitled "ADSM AFS/DFS Backup Clients Version 2.1" (IBM Document SH26-4048-00, International Business Machines, copyright 1996) and "ADSM Concepts" (IBM Document SG24-4877--00, International Business Machines, copyright 1997) describe the use of the ADSM software in an AFS/DFS distributed file system. These publications are incorporated herein by reference in their entirety.

Network traffic can be significantly increased if the AFS/DFS server and backup server are in one physical location, i.e., San Jose, Calif., and the AFS/DFS client and backup client requesting to backup a file in the AFS/DFS server are in a distant geographical location, i.e., Tucson, Ariz. If a user in Tucson wanted to backup a file that resided in the global name space managed by the AFS/DFS server in San Jose, prior art client/server protocol would have the AFS/DFS client in Tucson read the file, which requires transmittal of the file from San Jose to Tucson over the network, and then send the file back to the backup server in San Jose for backup storage. Such network traffic problems are exasperated when the client requesting the backup is separated by a long geographical distance from the server.

SUMMARY OF THE INVENTION

To address the shortcomings in the prior art described above, preferred embodiments of the present invention provide a system for backing up files in a distributed computing system. A file server maintains files in a shared name space. The file server provides a first backup client program and a second backup client program with access to the files in the shared name space. The first backup client program initiates a backup request to backup a requested file. A determination is made as to whether the requested file is maintained in the shared name space. The backup request is transmitted to the second backup client program upon determining that the requested file is maintained in the shared name space. The second backup client program transmits a message to the file server to provide the requested file. The file server transmits the requested file with the file server to the second backup client program. The second backup client program then transmits the requested file to a backup server program. The backup server program stores the requested file in a storage device.

In further embodiments, the first backup client program is on a first computer machine, the second backup client program is on a second computer machine, the backup server program is on a third computer machine, and the file server is on a fourth computer machine. The first computer, second computer, third computer, and fourth computer communicate over a network system.

In yet further embodiments, the first backup client program is on a first computer machine, the second backup client program and backup server program are on a second

computer machine, and the file server program is on a third computer machine. The first computer machine, second computer machine, and third computer machine communicate over a network system.

It is an object of preferred embodiments of the present invention to provide a system for backing up files in a shared name space maintained in a file server which is part of a distributed computing environment on a storage device managed by a backup server program, such as a hierarchical storage management program.

It is yet a further object to reduce network traffic throughout the distributed computing environment by having a proxy client including a copy of the backup client program read the file from the file server and transmit the file to the backup server to store in a storage device. In this way, a client at a remote location backing up a file does not read a file and retransmit such file back to the location from where the file came.

It is still a further object that data throughput rates be increased between the backup client program and the backup server program when the backup client program transmits files to the backup server program.

It is yet a further object that authentication be provided in using a proxy client to access files in the file server.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. **1** is a block diagram illustrating a software and hardware environment in which preferred embodiments of the present invention are implemented;

FIG. **2** is a block diagram illustrating an alternative software and hardware environment in which preferred embodiments of the present invention are implemented; and

FIG. **3** is a flowchart showing logic to retrieve and backup data in accordance with preferred embodiments of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

In the following description, reference is made to the accompanying drawings which form a part hereof, and which is shown, by way of illustration, several embodiments of the present invention. It is understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

Hardware and Software Environment

FIGS. **1** and **2** illustrate hardware and software environments in which preferred embodiments of the present invention are implemented. FIG. **1** illustrates a distributed computing system **2** comprised of four separate computing machines: a client **4**, a backup server **6**, a file server **8**, and a proxy client **10**. These four computing machines may be comprised of a personal computer, workstation, mainframe, etc. The computers **4, 6, 8, 10** would include an operating system such as AIX, OS/2, Unix, Microsoft Windows, etc. These four machines **4, 6, 8, 10** include software to allow the machines to function as components in a distributed computing system **2**, such as the IBM or Transarc Distributed Computing Environment (DCE) products. These computer machines **4, 6, 8, 10** may communicate via any suitable network technology known in the art, such as LAN, WAN, SNA networks, TCP/IP, the Internet, etc.

In the embodiment of FIG. **1**, the backup server **6**, storage devices **22**, proxy client **10**, and file server **8** are located in a Location A and the client **4** is in a Location B. Location A and B may be in distant geographical locations. In alternative embodiments, the file server **8**, backup server **6**, proxy client **10**, and client **4** can be in a single location, dispersed throughout a single site, dispersed throughout different sites in the same geographical proximity, dispersed throughout different sites at distant geographical locations, etc. If the proxy client **10** and backup server **6** are on separate machines, then a high-speed connection line **12**, e.g., a HIPPI or a high speed switch, such as the high speed switch built into the SP2 architecture, could connect the proxy client **10** and backup server **6**.

The client **4** and proxy client **10** include a distributed file system (DFS) client program **14** that provides communication with the file server **8** and access to files in a shared name space. The file server **8** includes a DFS server program **16** that manages the shared name space and makes data in the shared name space available to machines within the distributed computing system **2** running the DFS client program **14**. The DFS server program **16** further runs various distributed file system management processes. The DFS server program **16** and client program **14** may be part of a distributed file system (DFS) such as the AFS and DFS systems available from Transarc, the IBM Distributed Computing Environment (DCE), the Network File Server ("NFS") products from Sun Microsystems, Inc, or any other suitable distributed file system software known in the art. The terms "DFS client program" and "DFS server program," as used herein, refer generally to a DFS system and not to the particular DFS system provided by Transarc, IBM, or any other software provider. The DFS client program **14** and server program **16** include a communication protocol that allows the client **4** and proxy client **10**, including the DFS client program **14**, to interface with the file server **8** via the DFS server program **16**. In preferred embodiments, the DFS client program **14** and server program **16** may include a protocol, such as the DCE Remote Procedure Call (RPC), to provide communication therebetween. However, those skilled in the art will appreciate that alternative DFS communication protocols could be used to provide communication among systems within the distributed computing environment.

Machines running the DFS client program **14** are capable of accessing files in the shared name space managed by the DFS server program **16**, regardless of where those files are physically located. The files would conform to a uniform global name space, providing attached machines **4, 6, 8, 10** with a global view of a set of files and directories independent of machine boundaries. The client **4**, backup server **6**, file server **8**, and proxy client **10** may access the same shared name space and use the same global naming system in the distributed computing system **2**. This allows access to the shared name space regardless of where the client **4**, backup server **6**, file server **8**, and proxy client **10** are located.

The file server **8** or some other machine could perform authentication services to allow clients, such as client **4** and proxy client **10**, to access files in a file server **8**. In preferred embodiments, the DCE/RPC authentication protocols are implemented in the DFS client **14** and server **16** programs. Under such authentication protocols, when a user at a client **4** logs in, the client **4** requests a ticket to provide the client **4** access to a set of files maintained by the file server **8**. To access a file in the shared name space, the client **4** or proxy client **10** establishes communication with the DFS server program **16** in the file server **8** using the RPC protocol. Part of this protocol would require the client **4** or proxy client **10** to present the authentication ticket to the DFS server pro-

gram **16**, which would determine whether the requesting client **4**, **10** can access the files requested in the shared name space.

In preferred embodiments, the client **4** may establish communication with the proxy client **10** via an RPC call between the DFS client programs **14** in the client **4** and proxy client **10**. The client **4** could transfer its authentication ticket to the proxy client **10** through the RPC protocol. The proxy client **10** could, in turn, establish communication with the file server **8** via an RPC call established between the DFS client program **18** in the proxy client **10** and the DFS server program **16** in the file server **8**. Once communication is established, the proxy client **10** could use the authentication ticket provided by the client **4** to access files in the shared name space. In this way, the proxy client's **10** level of access to the file server **8** is limited to the level of access provided the client **4** because the proxy client **10** uses the authentication ticket from the client **4** to access the shared name space.

The client **4** includes a backup client program **18** that allows the client **4** to communicate with the backup server **4** to backup data to which the client **4** has access. The backup client program **18** may be comprised of any program that allows a client to communicate with a server to backup and archive data, such as the IBM ADSM client. The backup server **6** includes a backup server program **20** that stores and manages data in storage devices **22**. The storage devices **22** may be comprised of any non-volatile memory device suitable for long term storage of data, such as a tape library, optical disk library, hard disk drives, holographic units, etc. The backup server program **20** may include a database program to manage and track the location of data in the storage devices **22**. The backup server program **20** further includes communication protocol software to communicate with the backup client program **18**. The backup server program **20** may be comprised of any program that allows a server to manage and backup data in an attached storage device **22**, such as the IBM ADSM server program.

In preferred embodiments the backup client program **18** may transfer a backup request to another backup client program **18**. For instance, the backup client program **18** in the client **4** may transfer a request to backup a file in the file server **8** to the backup client **18** in the proxy client **10**.

FIG. **2** illustrates an alternative embodiment in which the proxy client **24** is a single computer machine including the backup server program **20**, the DFS client program **14**, and the backup client program **18**. The proxy client computer **24** may be a personal computer, workstation, mainframe, etc. The proxy client computer **24** and file server **8** may be in the same geographical location and/or site, and the client **4** may be at another site and/or distant geographical location. In the embodiment of FIG. **2** where the backup client program **18** and backup server program **20** are located on the same computer machine (node) **24**, the backup client **18** and server **20** programs may communicate using the memory of the proxy client computer **24**. For instance, the IBM ADSM product provides a shared memory protocol for transferring data between an ADSM client and server located on the same machine using a common memory area on the computer machine. The backup client program **18** would access and read data and transmit such data through the shared memory space to the backup server program **20**. The backup server program **20** would read the data copied to the shared memory space and then manage the storage of the transmitted data to the storage devices **22**. As with the embodiment in FIG. **1**, the backup client program **18** and backup server program **20** share the same global file name space through the DFS client program **14** which provides access to the file server **8** and files maintained therein. The proxy client computer **24** executes the DFS client program **14** to interface with the DFS server program **16** and access files in the shared name space maintained in the file server **8**.

Thus, the preferred embodiments may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass one or more computer programs and data files accessible from one or more computer-readable devices, carriers, or media, such as a magnetic storage media, "floppy disk," CD-ROM, a file server providing access to the programs via a network transmission line, holographic unit, etc. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Using the Proxy Client to Backup Data

FIG. **3** is a flowchart illustrating logic implemented in the programs **14**, **16**, **18**, **20** described in FIGS. **1** and **2** to back-up files maintained in the file server **8** in storage devices **22** managed by the backup server program **20**. Those skilled in the art will recognize that this logic is provided for illustrative purposes only and that different logic may be used to accomplish the same results.

Control begins at block **40** which represents the client **4** initiating a backup operation of a file with the backup client program **18**. The term "file" as used herein refers to an entire volume, logical unit, directory, subdirectory, individual file or any other image of data. Control transfers to block **42** which is a decision block representing the backup client program **18** and/or DFS client program **14** determining whether the file to be backed up is in the shared name space. In the AFS file system, including "/afs" in the file path typically indicates that a file is in a shared name space. In the DFS file system, including "/ . . . " typically indicates that a file is in a shared name space. If the requested file is in the shared name space then control transfers to block **44**; otherwise, control transfers to block **46**. If the file is not in the shared name space, then at block **46**, the backup client program **18** in the client **4** accesses the file and reads the file. Control transfers to block **48** which represents the backup client program **18** in the client **4** transmitting the accessed file to the backup server **6**. At block **50**, the backup server, operating under control of the backup server program **20**, backs up the file in the storage devices **22**.

If the file is in the shared name space, then, at block **44**, the client **4**, operating under control of the DFS client program **14**, makes a call to the proxy client **10** to perform the backup operation. The client **4** may use the RPC protocol to interface with the proxy client **10**. In side the proxy client **10**, the backup request is passed to the backup client program **18**. Control transfers to block **52** which represents the client **4**, using the DCE/RPC protocol, passing an authentication ticket presented to the client **4**. The client **4** may include the authentication ticket with the request to the proxy client **10** to backup the file. This authentication ticket determines the level of access the client **4** has to the file server **8**. Control transfers to block **54** which represents the proxy client **10** making a call to the file server **8** to access the file to be backed up. In the preferred embodiments, the proxy client **10**, under control of the DFS client **14**, makes a RPC call to the DFS server program **16** in the file server **8** to

access the shared name space. Control transfers to block **56** which represents the proxy client **10**, through the RPC call established with the DFS client **14** and server **6** programs, providing the authentication ticket passed from the client **4** to the file server **8**. The proxy client **10** may include the authentication ticket with the call to the file server **8** to access the file.

Control proceeds to block **58** which is a decision block representing the DFS server program **16** in the file server **8** determining whether the authentication ticket permits the proxy client **10** to access the file to be backed up. If so, control transfers to block **60**; otherwise, control transfers to block **62**. If the authentication ticket does not permit access to the file to be backed up, then control proceeds to block **62** which represents the file server **8** sending a message to the proxy client **10** that access to the file is not permitted. At block **64**, the proxy client **10** notifies the client **4** that access to the file to be backed up was denied. As discussed, in preferred embodiments, the client **4** and proxy client **10** can communicate using the RPC interface.

If the authentication ticket from the client **4** permits access to the file, then at block **60**, the file server **8** provides the requested file to the proxy client **10** via the call established between the DFS client program **14** in the proxy client **10** and the DFS server program **16** in the file server **8**. Control proceeds to block **68** which represents the backup client program **18** in the proxy client **10** transmitting the file provided from the file server **8** to the backup server program **20** in the backup server **6**. As discussed, the backup client program **18** in the proxy client **10** may communicate with the backup server program **20** via the high speed communication line **12**. Control transfers to block **70** which represents the backup server program **20** transferring the file to the storage devices **22**. In preferred embodiments, the backup server program **20** transfers data from the proxy client **10** the storage devices **22** simultaneously as the data is transmitted from the proxy client **10**. Control transfers to block **72** which represents the proxy client **10** providing status information to the client **4** upon completion of the file backup. Control then proceeds to block **74** which represents the client **4** providing the status information to the user at client **4**.

If authentication was unnecessary, then the logic of FIG. **3** would not include the steps to authenticate the level access for the proxy client **10**. In such case, the file server **8** would provide the requested file to the proxy client **10** without performing authentication verification to determine if access is permitted to the requested file.

The logic described in FIG. **3** is implemented in a distributed computing environment **2** in which the proxy client **10**, including the backup client program **18**, and the backup server program **20** are on a separate computer machines **6**, **10**, such as in the environment described in FIG. **1**. In the alternative embodiment of FIG. **2**, the backup client program **18** and the backup server program **20** are installed on the same proxy client computer **24**. Thus, the proxy client computer **24** includes the backup server program **20**. In such case, at block **68**, the backup client program **18** would transfer the file to be backed up to the backup server program **20** via a shared memory space and not the high speed transmission line **12** as is the case with FIG. **1**. In alternative embodiments, the backup client program **18** and backup server program **20** may communicate via the network system providing communication among the devices in the distributed computing system **2**.

With the preferred embodiments discussed above, network traffic is significantly reduced because the client **4**, which may be in a distant geographical location, e.g., Location B, from where the backup server **6** and file server **8** are located, e.g., Location A, does not have to read the file from the file server **8** and then retransmit the file back to the backup server **6**. Greater reductions in network traffic are further realized if the backup server **6**, proxy client **10** and file server **8** are in a proximate location. Network traffic may be further reduced in those embodiments which utilize a separate high speed connection **12** (FIG. **1**) between the proxy client **10** and the backup server **6** to allow the backup client program **18** in the proxy client **10** to transfer the file to the backup server program **20** at higher data rates and bypass the main communication lines of the network.

Conclusion

This concludes the description of the preferred embodiments of the invention. The following describes some alternative embodiments for accomplishing the present invention.

Preferred embodiments utilize current available products, such as ADSM, DFS, AFS, and NFS. However, any suitable program capable of performing the functions described herein could be substituted for the preferred embodiments described herein.

In preferred embodiments, certain operations are described as being performed by certain computer programs **14**, **16**, **18**, **20**. However, those skilled in the art will appreciate that an alternative combination of programs could be used to implement the logic of preferred embodiments of the invention. Moreover the programs **14**, **16**, **18**, **20** may themselves be comprised of one or more component computer programs, e.g., executable and data files, that function together to perform the operations described with respect to programs **14**, **16**, **18**, **20**.

In summary, preferred embodiments disclose a system for backing up files in a distributed computing system. A file server maintains files in a shared name space. The file server provides a first backup client program and a second backup client program with access to the files in the shared name space. The first backup client program initiates a backup request to backup a requested file. A determination is made as to whether the requested file is maintained in a shared name space. The backup request is transmitted to the second backup client program upon determining that the requested file is maintained in the shared name space. The second backup client program transmits a message to the file server to provide the requested file. The file server transmits the requested file with the file server to the second backup client program. The second backup client program then transmits the requested file to a backup server program. The backup server program stores the requested file in a storage device.

The foregoing description of the preferred embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

What is claimed is:

1. A method for backing up files in a distributed computing system, comprising the steps of:

maintaining, with a file server, files in a shared name space, wherein a first backup client program and a second backup client program are capable of accessing files in the shared name space via the file server;

initiating a backup request with the first backup client program to backup a requested file;

determining whether the requested file is maintained in the shared name space,

transmitting the backup request from the first backup client program to the second backup client program upon determining that the requested file is maintained in the shared name space;

transmitting a message with the second backup client program to the file server to provide the requested file;

transmitting the requested file with the file server to the second backup client program;

transmitting with the second backup client program the requested file to a backup server program; and

storing with the backup server program the requested file in a storage device.

2. The method of claim 1, wherein the first backup client program is on a first computer machine, the second backup client program is on a second computer machine, the backup server program is on a third computer machine, and the file server is on a fourth computer machine, wherein the first computer machine, second computer machine, third computer machine, and fourth computer machine communicate over a network system.

3. The method of claim 2, wherein the second backup client program and the backup server program communicate via a high speed communication line connecting the second computer machine and the third computer machine.

4. The method of claim 2, wherein the first and second computer machines include a distributed file system (DFS) client program and wherein the file server includes a DFS server program, wherein the DFS client program and DFS server program interface the first and second computer machines with the file server to allow access to files in the shared name space maintained by the file server.

5. The method of claim 2, further including the steps of:

issuing an authentication ticket to the first computer machine providing access to a set of files in the shared name space;

transmitting with the first computer machine the authentication ticket to the second computer machine;

transmitting with the second computer machine the authentication ticket to the file server in the fourth computer machine; and

determining with the file server whether the requested file is in the set of files to which the authentication ticket permits access, wherein the step of transmitting the requested file with the file server to the second computer machine including the second backup client program occurs upon determining that the requested file is in the set of files to which the authentication ticket permits access.

6. The method of claim 1, wherein the first backup client program is on a first computer machine, the second backup client program and backup server program are on a second computer machine, and the file server program is on a third computer machine, wherein the first computer machine, second computer machine, and third computer machine communicate over a network system.

7. The method of claim 6, wherein the second backup client program and the backup server program communicate via a shared memory within the second computer machine.

8. The method of claim 6, wherein the first backup client program communicates the request to backup the file to the second backup client program, wherein the first and second computer machines include a distributed file system (DFS) client program, wherein the file server includes a DFS server program, further including the step of the DFS client program in the second computer machine interfacing with the DFS server program in the third computer machine to provide the second computer machine access to files in the shared name space maintained by the file server.

9. A distributed computing system for backing up files in a shared name space, comprising:

(a) a first backup client program, including means for initiating a backup request to backup a requested file;

(b) a second backup client program;

(c) a backup server program;

(d) a storage device managed by the backup server program;

(e) a file server, wherein the file server provides access to files included in a shared name space, wherein the first backup client program and the second backup client program have access to files maintained in the shared name space through the file server;

(f) means for determining whether the requested file is included in the shared name space;

(g) means for transmitting the backup request to the second backup client program upon determining that the requested file is included in the shared name space;

(h) means for transmitting a message with the second backup client program to the file server to provide the requested file;

(i) means, performed by the file server, for transmitting the requested file to the second backup client program;

(j) means, performed by the second backup client program, for transmitting the requested file to the backup server program; and

(k) means, performed by the backup server program, for storing the requested file in the storage device.

10. The distributed computing system of claim 9, further including:

a first computer machine including the first backup client program;

a second computer machine including the second backup client program;

a third computer machine including the backup server program;

a fourth computer machine including the file server; and

a network system providing communication among the first computer machine, second computer machine, third computer machine, and fourth computer machine.

11. The distributed computing system of claim 10, further including a high speed communication line connecting the second computer machine and the third computer machine, wherein the second backup client program and the backup server program communicate via the high speed communication line.

12. The distributed computing system of claim 10, further including:

a distributed file system (DFS) client program included within the first and second computer machines; and

a DFS server program included in the file server, wherein the DFS client program and DFS server program inter-

faces the first and second computer machines and the file server to allow access to files in the shared name space maintained by the file server.

13. The distributed computing system of claim **10**, further including:

means for issuing an authentication ticket to the first computer machine providing access to a set of files in the shared name space;

means, performed by the first computer machine, for transmitting the authentication ticket to the second computer machine;

means, performed by the second computer machine, for transmitting the authentication ticket to the file server in the fourth computer machine; and

means, performed by the file server, for determining whether the requested file is in the set of files to which the authentication ticket permits access.

14. The distributed computing system of claim **9**, further including:

a first computer machine including the first backup client program;

a second computer machine including the second backup client program and the backup server program;

a third computer machine including the file server program;

a network system providing communication between the first computer machine, the second computer machine, and the third computer machine.

15. The distributed computing system of claim **14**, further including a shared memory within the second computer machine, wherein the second backup client program and the backup server program communicate via the shared memory.

16. The distributed computing system of claim **14**, further including:

a distributed file system (DFS) client program in the second computer machine;

a DFS server program included in the file server, wherein the DFS client program interfaces the second computer machine with the DFS server program to access files in the shared name space maintained by the file server.

17. An article of manufacture for use in programming a distributed computing system including a file server maintaining files in a shared name space, a first backup client program and a second backup client program, wherein the first and second backup client programs are capable of accessing files in the shared name space via the file server, the article of manufacture comprising at least one computer readable storage device including at least one computer program embedded therein that causes components within the distributed computing system to perform the steps of:

(a) initiating a backup request with the first backup client program to backup a requested file;

(b) determining whether the requested file is maintained in the shared name space;

(c) transmitting the backup request to the second backup client program upon determining that the requested file is maintained in the shared name space;

(d) transmitting a message with the second backup client program to the file server to provide the requested file;

(e) transmitting the requested file with the file server to the second backup client program;

(f) transmitting with the second backup client program the requested file to a backup server program; and

(g) storing with the backup server program the requested file in a storage device.

18. The article of manufacture of claim **17**, wherein the first backup client program is on a first computer machine, the second backup client program is on a second computer machine, the backup server program is on a third computer machine, and the file server is on a fourth computer machine, wherein the first computer machine, second computer machine, third computer machine, and fourth computer machine communicate over a network system.

19. The article of manufacture of claim **18**, wherein the second backup client program and the backup server program communicate via a high speed communication line connecting the second computer machine and the third computer machine.

20. The article of manufacture of claim **18**, wherein the second computer machine includes a distributed file system (DFS) client program and wherein the file server includes a DFS server program, wherein the DFS client program and DFS server program interface the second computer machine with the file server to allow access to files in the shared name space maintained by the file server.

21. The article of manufacture of claim **18**, further including the steps of:

issuing an authentication ticket to the first computer machine providing access to a set of files in the shared name space;

transmitting with the first computer machine the authentication ticket to the second computer machine;

transmitting with the second computer machine the authentication ticket to the file server in the fourth computer machine;

determining with the file server whether the requested file is in the set of files to which the authentication ticket permits access, wherein the step of transmitting the requested file with the file server to the second computer machine including the second backup client program occurs upon determining that the requested file is in the set of files to which the authentication ticket permits access.

22. The article of manufacture of claim **17**, wherein the first backup client program is on a first computer machine, the second backup client program and backup server program are on a second computer machine, and the file server program is on a third computer machine, wherein the first computer machine, second computer machine, and third computer machine communicate over a network system.

23. The article of manufacture of claim **22**, wherein the second backup client program and the backup server program communicate via a shared memory within the second computer machine.

24. The article of manufacture of claim **22**, wherein the first computer machine communicates the request to backup the file from the first backup client program to the second backup client program in the second computer machine, wherein the second computer machine includes a distributed file system (DFS) client program, and wherein the file server includes a DFS server program, wherein the DFS client program interfaces the second computer machine with the DFS server program to provide the second computer machine access to files in the shared name space maintained by the file server.

* * * * *

US006119128A

United States Patent [19]

Courter et al.

[11] Patent Number: 6,119,128

[45] Date of Patent: Sep. 12, 2000

[54] **RECOVERING DIFFERENT TYPES OF OBJECTS WITH ONE PASS OF THE LOG**

[75] Inventors: **Daniel Keith Courter**, Antioch; **Ming-Hung Hu**, San Jose; **Laura Michiko Kunioka-Weis**, Morgan Hill; **Thomas Majithia**; **Deborah A. Matamoros**, both of San Jose; **James Alan Ruddy**, Gilroy; **Yufen Wang**, Saratoga, all of Calif.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **09/050,554**

[22] Filed: **Mar. 30, 1998**

[51] **Int. Cl.**[7] **G06F 12/00**

[52] **U.S. Cl.** **707/202**; 707/200; 707/201; 707/203; 707/204; 707/205

[58] **Field of Search** 707/200, 201, 707/202, 203, 204, 205

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,945,474	7/1990	Elliott et al.	714/16
5,276,872	1/1994	Lomet et al.	707/202
5,278,982	1/1994	Daniels et al.	707/202
5,280,611	1/1994	Mohan et al.	707/8
5,327,532	7/1994	Ainsworth et al.	709/248
5,333,303	7/1994	Mohan	714/20
5,455,944	10/1995	Haderle et al.	707/202
5,455,946	10/1995	Mohan et al.	707/202
5,561,795	10/1996	Sarkar	707/202
5,561,798	10/1996	Haderle et al.	707/202
5,574,897	11/1996	Hermsmeier et al.	707/1
5,581,750	12/1996	Haderle et al.	707/202
5,625,820	4/1997	Hermsmeier et al.	707/202
5,721,918	2/1998	Nilsoson	707/202
5,832,508	11/1998	Sheman	707/200
5,873,096	2/1999	Lim	707/201
5,903,898	5/1999	Cohen	707/204
5,907,848	5/1999	Zaiken	707/202
5,920,873	7/1999	Van Huben	707/202
5,926,816	7/1999	Bauer	707/8

OTHER PUBLICATIONS

"Incremental Data Base Log Image Copy", IBM Technical Disclosure Bulletin, vol. 25, No. 7B, pp. 3730–3732, 1982.

"Technique For Data Recovery excluding Portions of The Log Without Requiring a Full Image Copy", IBM Technical Disclosure Bulletin, vol. 36, No. 11, pp. 359–360, Nov. 1993.

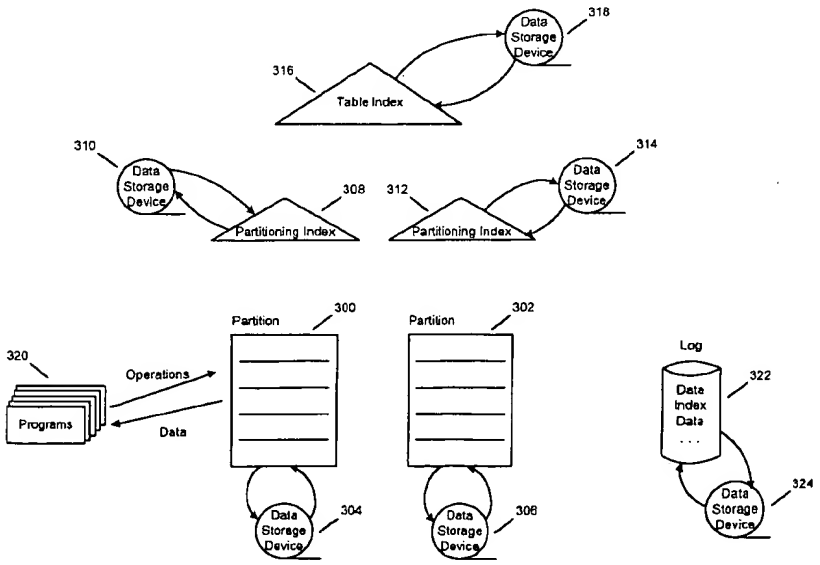
Fernando de Ferreira Rezende, et al., "Employing Object-Based LSNs in A Recovery Strategy", Database And Expert Systems Applications, 7th International Conference, DEXA '96, pp. 116–129, Sep. 9–13, 1996.

Primary Examiner—Thomas G. Black
Assistant Examiner—Diane D. Mizrahi
Attorney, Agent, or Firm—Pretty, Schroeder & Poplawski

[57] **ABSTRACT**

A method, apparatus, and article of manufacture for a computer implemented recovery system for restoring a database in a computer. The database contains objects and is stored on a primary data storage device connected to the computer. Objects of different types in the database are copied from the primary data storage device to a secondary data storage device. Modifications to the objects are logged in a log file. A recovery indicator is received that indicates that recovery of the objects in the database is required. The objects are copied from the secondary data storage device to the database on the primary data storage device. Modifications in the log file are applied to the copied objects during one pass through the log file.

24 Claims, 6 Drawing Sheets



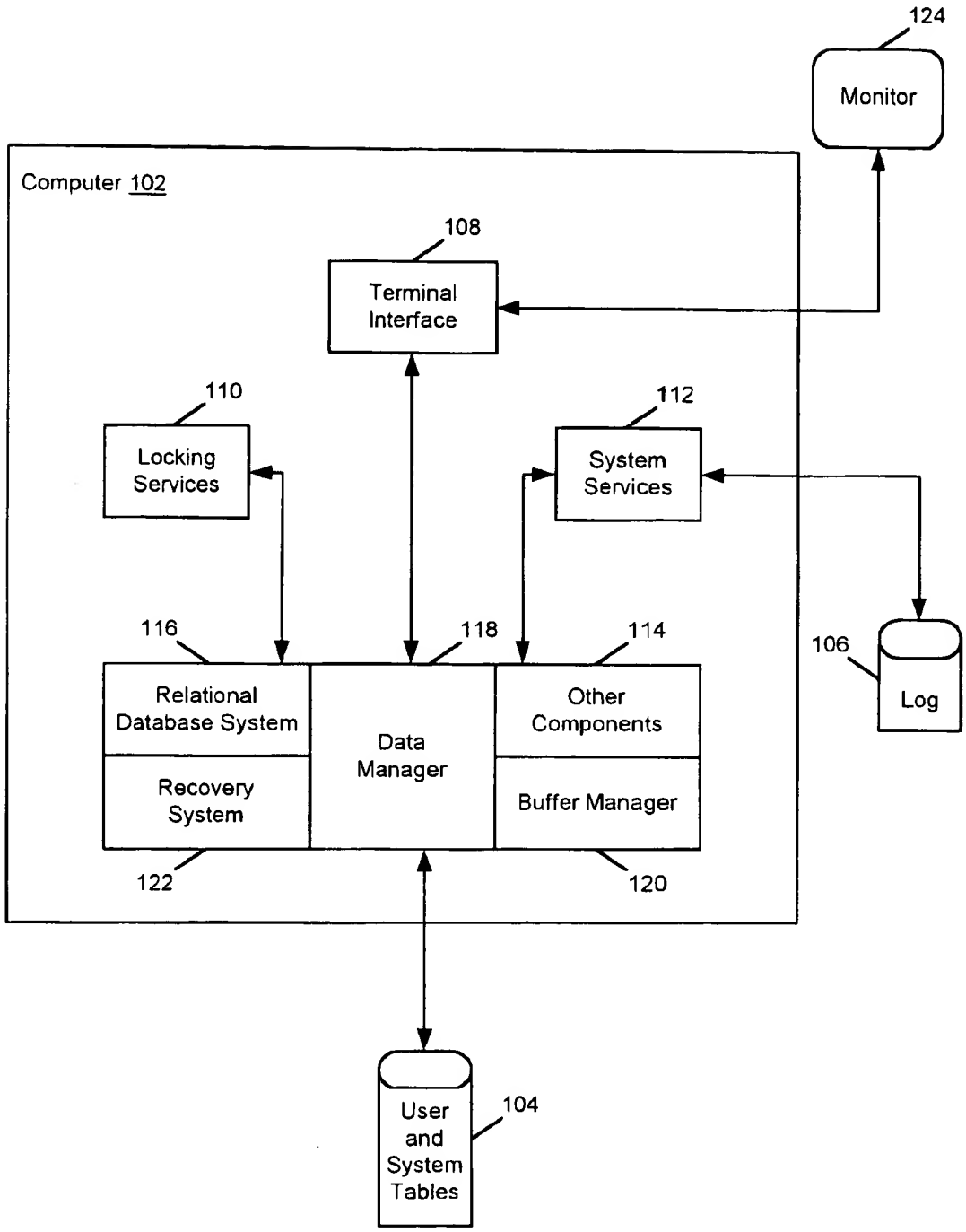

124
Monitor
Computer 102
108
Terminal Interface
110
Locking Services
112
System Services
116
118
114
Relational Database System
Recovery System
Data Manager
Other Components
Buffer Manager
106
Log
122
120
User and System Tables
104

FIG. 1

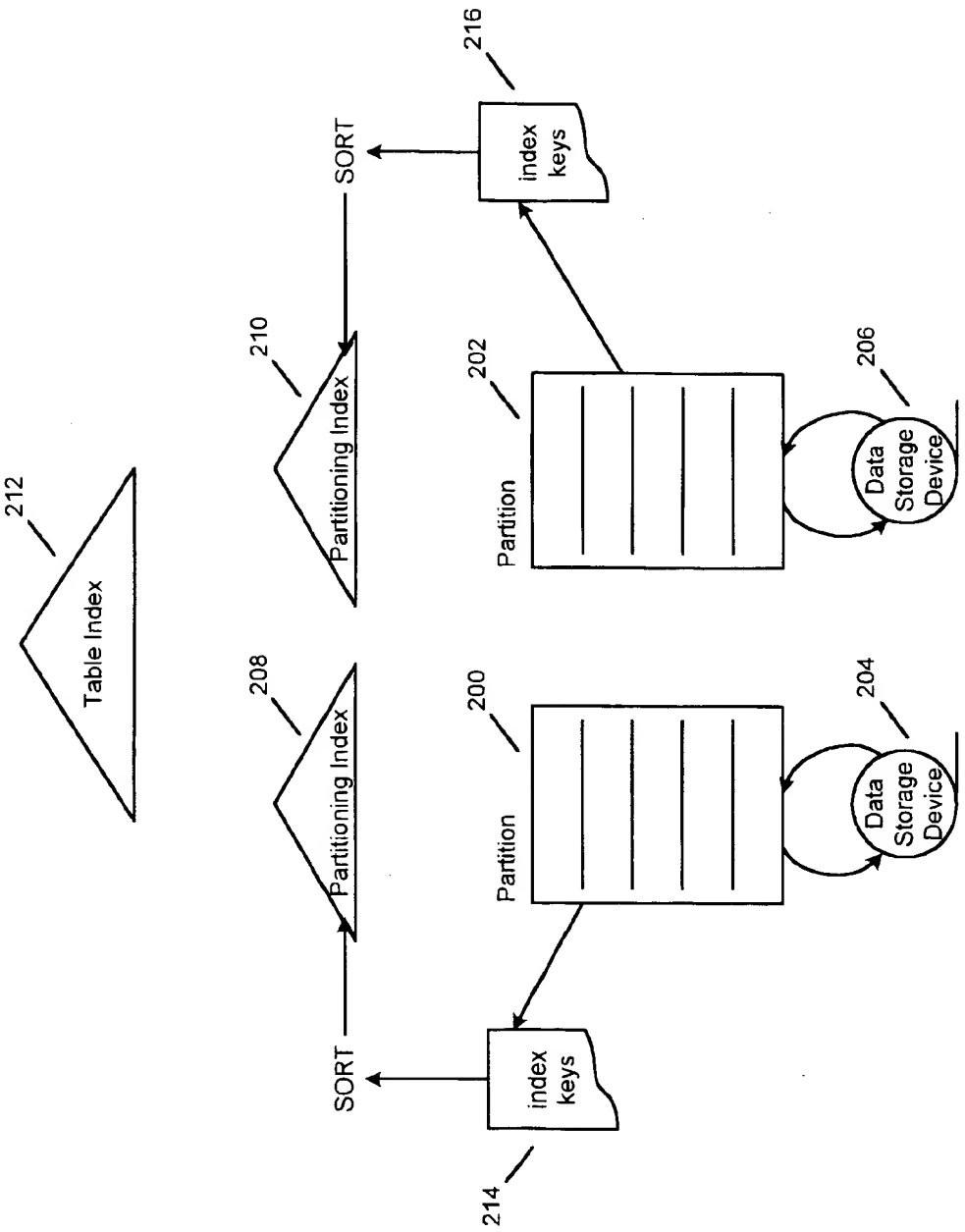

212
Table Index
210
Partitioning Index
208
Partitioning Index
SORT
SORT
216
index keys
214
index keys
202
Partition
200
Partition
206
Data Storage Device
204
Data Storage Device

FIG. 2

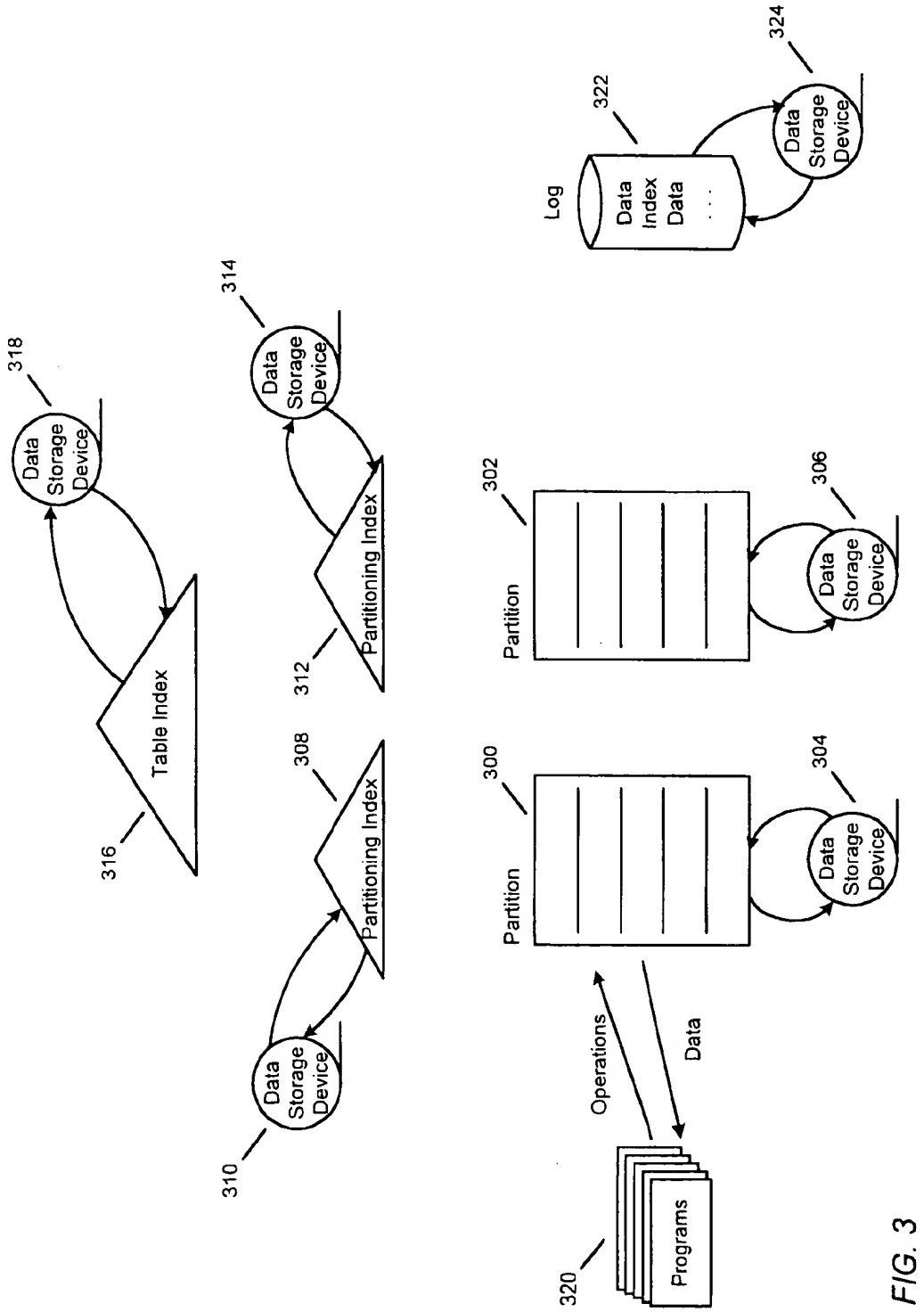

Log
Data
Index
Data
. . .
322
324
Data Storage Device
Partition
302
306
300
304
308
310
312
314
316
318
Table Index
Partitioning Index
Operations
Data
Programs
320

FIG. 3

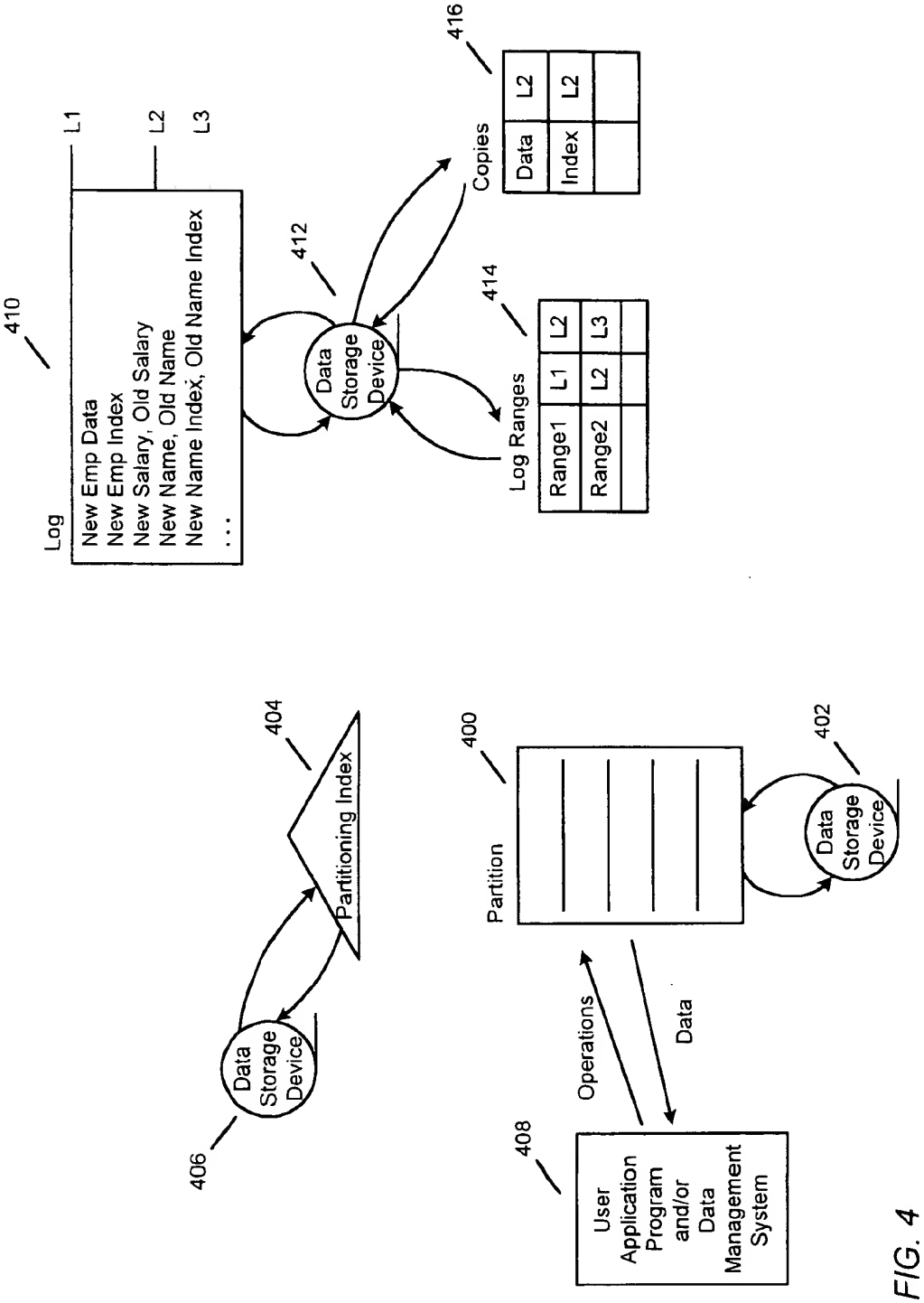

410
Log
New Emp Data
New Emp Index
New Salary, Old Salary
New Name, Old Name
New Name Index, Old Name Index
...
L1
L2
L3
412
Data Storage Device
414
Log Ranges
Range1 L1 L2
Range2 L2 L3
416
Copies
Data L2 L2
Index
404
Partitioning Index
406
Data Storage Device
400
Partition
402
Data Storage Device
Operations
Data
408
User Application Program and/or Data Management System

FIG. 4

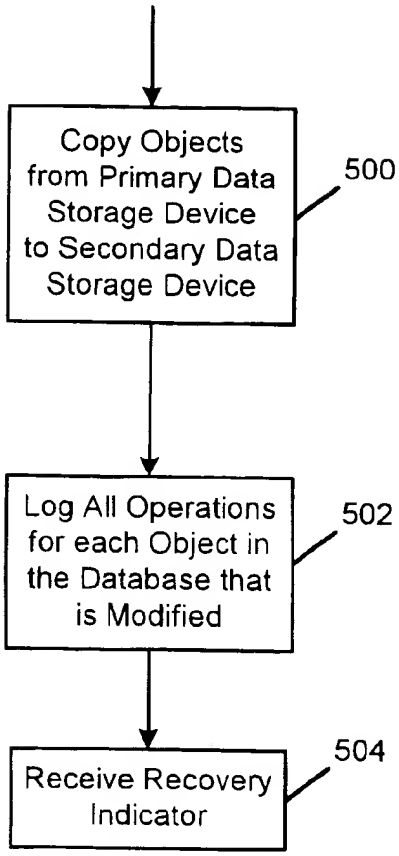

Copy Objects from Primary Data Storage Device to Secondary Data Storage Device
500
Log All Operations for each Object in the Database that is Modified
502
Receive Recovery Indicator
504

FIG. 5

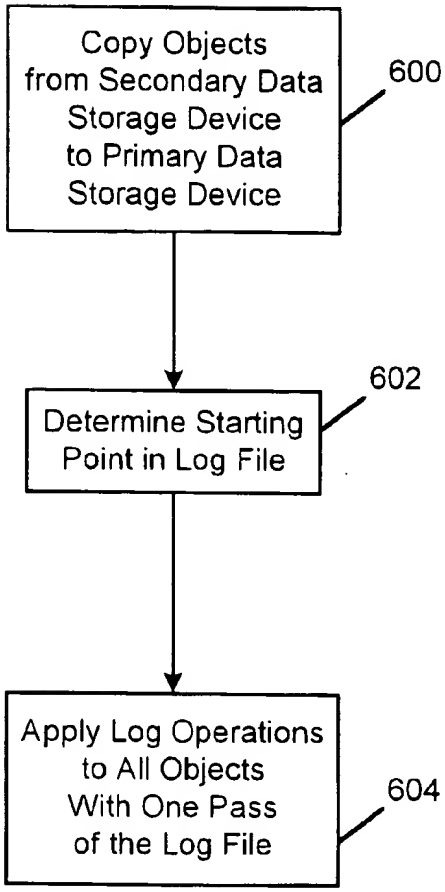

Copy Objects from Secondary Data Storage Device to Primary Data Storage Device
600
Determine Starting Point in Log File
602
Apply Log Operations to All Objects With One Pass of the Log File
604

FIG. 6

RECOVERING DIFFERENT TYPES OF OBJECTS WITH ONE PASS OF THE LOG

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates in general to computer-implemented database systems, and, in particular, to recovering different types of objects with one pass of the log.

2. Description of Related Art

Databases are computerized information storage and retrieval systems. A Relational Database Management System (RDBMS) is a database management system (DBMS) which uses relational techniques for storing and retrieving data. Relational databases are organized into tables which consist of rows and columns of data. The rows are formally called tuples. A database will typically have many tables and each table will typically have multiple tuples and multiple columns. The tables are typically stored on direct access storage devices (DASD), such as magnetic or optical disk drives for semipermanent storage.

A table is assigned to a tablespace. The tablespace contains one or more datasets. In this way, the data from a table is assigned to physical storage on DASD. Each tablespace is physically divided into equal units called pages. The size of the tablespace's pages is based on the page size of the bufferpool specified in the tablespace's creation statement. The bufferpool is an area of virtual storage that is used to store data temporarily. A tablespace can be partitioned, in which case a table may be divided among the tablespace's partitions, with each partition stored as a separate dataset. Partitions are typically used for very large tables.

A table may have an index. An index is an ordered set of pointers to the data in the table. There is one physical order to the rows in a table that is determined by the RDBMS software, and not by a user. Therefore, it may be difficult to locate a particular row in a table by scanning the table. A user creates an index on a table, and the index is based on one or more columns of the table. A partitioned table must have at least one index. The index is called the partitioning index and is used to define the scope of each partition and thereby assign rows of the table to their respective partitions. The partitioning indexes are created in addition to, rather than in place of, a table index. An index may be created as UNIQUE so that two rows can not be inserted into a table if doing so would result in two of the same index values. Also, an index may be created as a CLUSTERING index, in which case the index physically stores the rows in order according to the values in the columns specified as the clustering index (i.e., ascending or descending, as specified by the user).

RDBMS software using a Structured Query Language (SQL) interface is well known in the art. The SQL interface has evolved into a standard language for RDBMS software and has been adopted as such by both the American National Standards Institute (ANSI) and the International Standards Organization (ISO). The SQL interface allows users to formulate relational operations on the tables either interactively, in batch files, or embedded in host languages, such as C and COBOL. SQL allows the user to manipulate the data. As the data is being modified, all operations on the data are logged in a log file.

Typically, the database containing partitions and indexes is stored on a data storage device, called a primary data storage device. The partitions are periodically copied to another data storage device, called a secondary data storage device, for recovery purposes. In particular, the partitions stored on the primary data storage device may be corrupted, for example, due to a system failure during a flood, or a user may want to remove modifications to the data (i.e., back out the changes). In either case, for recovery, the partitions are typically copied from the secondary data storage device to the primary data storage device. Next, using the log file, the copied data is modified based on the operations in the log file. Then, the indexes are rebuilt. In particular, to rebuild the indexes, keys are copied from each row of each partition, sorted, and then used to create a partitioning index. Additionally, the table index is rebuilt via the same technique.

This technique for recovery of data and indexes is very costly in terms of performance. Additionally, users are not able to access data while recovery is taking place. For a user or company requiring the use of computers to do business, much money can be lost during recovery. Therefore, it is important to improve the efficiency of the recovery process, and there is a need in the art for an improved recovery technique.

SUMMARY OF THE INVENTION

To overcome the limitations in the prior art described above, and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention discloses a method, apparatus, and article of manufacture for a computer implemented recovery system for restoring a database in a computer.

In accordance with the present invention, the database contains objects and is stored on a primary data storage device connected to the computer. Objects of different types in the database are copied from the primary data storage device to a secondary data storage device. Modifications to the objects are logged in a log file. A recovery indicator is received that indicates that recovery of the objects in the database is required. The objects are copied from the secondary data storage device to the database on the primary data storage device. Modifications in the log file are applied to the copied objects during one pass through the log file.

An object of the invention is to provide an improved recovery system for a database. Another object of the invention is to provide recovery for partitions, partitioning indexes, and table indexes simultaneously. Yet another object of the invention is to provide a recovery system for a database that requires only one pass of a log file to apply modifications to the database.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 illustrates an exemplary computer hardware environment that could be used in accordance with the present invention;

FIG. 2 illustrates a conventional system for recovery of a database;

FIG. 3 illustrates the recovery system in accordance with the present invention;

FIG. 4 provides an example that illustrates the recovery system in accordance with the present invention;

FIG. 5 is a flow diagram illustrating the steps performed by the recovery system prior to recovery of a database in accordance with the present invention; and

FIG. 6 is a flow diagram illustrating the steps performed by the recovery system to recover a database in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following description of the preferred embodiment, reference is made to the accompanying drawings which form a part hereof, and which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized as structural changes may be made without departing from the scope of the present invention.

Hardware Environment

FIG. **1** illustrates an exemplary computer hardware environment that could be used in accordance with the present invention. In the exemplary environment, a computer system **102** is comprised of one or more processors connected to one or more data storage devices **104** and **106** that store one or more relational databases, such as a fixed or hard disk drive, a floppy disk drive, a CDROM drive, a tape drive, or other device.

Operators of the computer system **102** use a standard operator interface **108**, such as IMS/DB/DC®, CICS®, TSO®, OS/390® or other similar interface, to transmit electrical signals to and from the computer system **102** that represent commands for performing various search and retrieval functions, termed queries, against the databases. In the present invention, these queries conform to the Structured Query Language (SQL) standard, and invoke functions performed by Relational DataBase Management System (RDBMS) software. In the preferred embodiment of the present invention, the RDBMS software comprises the DB2® product offered by IBM for the MVS® or OS/390® operating systems. Those skilled in the art will recognize, however, that the present invention has application program to any RDBMS software that uses SQL.

As illustrated in FIG. **1**, the DB2® architecture for the MVS® operating system includes three major components: the Internal Resource Lock Manager (IRLM) **110**, the Systems Services module **112**, and the Database Services module **1 14**. The IRLM **110** handles locking services for the DB2® architecture, which treats data as a shared resource, thereby allowing any number of users to access the same data simultaneously. Thus concurrency control is required to isolate users and to maintain data integrity. The Systems Services module **112** controls the overall DB2® execution environment, including managing log data sets **106**, gathering statistics, handling startup and shutdown, and providing management support.

At the center of the DB2® architecture is the Database Services module **114**. The Database Services module **114** contains several submodules, including the Relational Database System (RDS) **116**, the Data Manager **118**, the Buffer Manager **120**, the Recovery System **122**, and other components **124**, such as an SQL compiler/interpreter. These submodules support the functions of the SQL language, i.e. definition, access control, interpretation, compilation, database retrieval, and update of user and system data. The Recovery System **122** works with the components of the computer system **102** to restore a database.

The present invention is generally implemented using SQL statements executed under the control of the Database Services module **114**. The Database Services module **114** retrieves or receives the SQL statements, wherein the SQL statements are generally stored in a text file on the data storage devices **104** and **106** or are interactively entered into the computer system **102** by an operator sitting at a monitor **124** via operator interface **108**. The Database Services module **114** then derives or synthesizes instructions from the SQL statements for execution by the computer system **102**.

Generally, the RDBMS software, the SQL statements, and the instructions derived therefrom, are all tangibly embodied in a computer-readable medium, e.g. one or more of the data storage devices **104** and **106**. Moreover, the RDBMS software, the SQL statements, and the instructions derived therefrom, are all comprised of instructions which, when read and executed by the computer system **102**, causes the computer system **102** to perform the steps necessary to implement and/or use the present invention. Under control of an operating system, the RDBMS software, the SQL statements, and the instructions derived therefrom, may be loaded from the data storage devices **104** and **106** into a memory of the computer system **102** for use during actual operations.

Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Those skilled in the art will recognize that the exemplary environment illustrated in FIG. **1** is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware environments may be used without departing from the scope of the present invention.

Recovering Different Types of Objects With One Pass of The Log

The present invention provides a recovery system **122** for recovering different types of objects using only one pass through a log file. In particular, table partitions of a database, along with indexes (e.g., partitioning indexes and table indexes), are copied to one or more data storage devices, such as magnetic tape. The database may be stored on a primary data storage device, while the copies of the database partitions and indexes are stored on a secondary data storage device. The primary and secondary data storage devices could be the same or different devices.

Then, as modifications are made to the data in the table partitions, the modifications are logged in a log file. If recovery of the table partitions and partitioning indexes are required, the recovery system **122** of the present invention copies the table partitions and partitioning indexes from the secondary data storage device back to the database. Then, the recovery system **122** modifies both the table partitions and the partitioning indexes while making one pass through the log file. That is, the recovery system **122** extracts all of the pertinent log records containing updates to all of the objects being recovered in a single read pass of logged changes.

The recovery system **122** allows for independent recovery of the data and indexes, and a significant decrease in elapsed time since the log file updates are done for all objects in the database with one pass through the log file.

FIG. **2** illustrates a conventional system for recovery of a database. In a conventional system, partitions **200** and **202** of a database are copied from primary data storage devices to secondary data storage devices **204** and **206**. In a conventional system, the partitioning indexes **208** and **210** and the table index **212** are not stored on secondary data storage devices. Then, when recovery is required, the conventional

system copies the partitions **200** and **202** from the secondary data storage devices **204** and **206** to the database on the primary data storage devices. The conventional system applies modifications logged in a log file to the copied partitions. Then, the conventional system reads each row of each partition **200** and **202** and retrieves index keys **214** and **216** for each row of each partition **200** and **202**. The index keys **214** and **216** are sorted and are used to rebuild indexes **208** and **210**, respectively. Table index **212** is rebuilt in the same manner. This procedure has a high performance cost.

FIG. **3** illustrates the recovery system **122** in accordance with the present invention. Initially, the partitions **300** and **302** are copied to secondary data storage devices **304** and **306**. Also, partitioning indexes **308** and **312** are copied to secondary data storage devices **310** and **314**. The table index **316** is also copied to a secondary data storage device **318**. Then, as application programs **320** modify the database by adding, updating, or deleting data via operations, the modifications are logged in the log file **322**. The log file may be copied to a secondary data storage device **324** if the log file on the primary storage device becomes full. The log file **322** contains information identifying modifications to both the partitions and indexes.

For recovery, the partitions from the data storage devices **304** and **306** are copied back to the primary data storage device. The partitioning indexes are copied from the secondary data storage devices **310** and **314** to the primary data storage device. Additionally, the table index is copied from the secondary data storage device **318** to the primary data storage device. Then, the log records subsequent to the last copying from the primary to the secondary data storage devices are applied to the partitions and indexes. In particular, while reading the log file through once, the recovery system **122** modifies both the partitions **300** and **302** and the indexes **308**, **312**, and **316**.

FIG. **4** provides an example that illustrates the recovery system **122** in accordance with the present invention. Initially, the partition **400** is copied to a secondary data storage device **402**, and the partitioning index **404** is copied to a secondary data storage device **406**. Then, a user application program and/or a data management system **408** perform operations on the data in the partition **400** and partitioning index **404**. The operations are logged in the log file **410**.

For example, if the first operation adds a new employee, the recovery system **122** modifies the partition **400** and the partitioning index **404**. In particular, the recovery system **122** adds entries to the log file for "New Emp Data" and "New Emp Index". Then, if an operation updates salary information for an employee, the recovery system **122** modifies the partition **400**. The log file then contains an entry for "New Salary, Old Salary" that identifies the salary before and after modification. Next, if the partition **400** and partitioning index **404** are to be copied, a log range file **414** and a copies file **416** are modified. In particular, the log file **410** is separated into ranges. The log range file **414** indicates each of the ranges, for example, that Range1 goes from range identifier L1 to range identifier L2. The copies file **416** indicates partitions and indexes that have been copied. The range identifier, for example, L2, indicates that the copied data for the partitions and indexes includes all of the operations logged up to range identifier L2.

Next, if the name of an employee is changed, the partition **400** and partitioning index **404** are modified. Then, the log file **410** contains an entry for "New Name, Old Name" that provides the new and old name of the employee whose name changed and an entry for "New Name Index, Old Name Index" that provides the index modification. Next, assuming that there is a loss of data, recovery of the data is required. Initially, the partition **400** and the partitioning index **404** are copied from secondary data storage devices back to the primary data storage device. Since, according to the copies file **416**, these copies include all modifications up to range identifier L2, only operations after range identifier L2 are applied to the partition **400** and the partitioning index **404** to recover the database. Moreover, during one pass through the log file, the recovery system **122** identifies the required modifications and applies them to the partition **400** and the partitioning index **404**.

FIG. **5** is a flow diagram illustrating the steps performed by the recovery system **122** prior to recovery of a database in accordance with the present invention. In Block **500**, the recovery system **122** copies objects from a primary data storage device to a secondary data storage device. In Block **502**, the recovery system **122** logs all operations for each object in the database that is modified. In Block **504**, the recovery system **122** receives a recovery indicator.

FIG. **6** is a flow diagram illustrating the steps performed by the recovery system **122** to recover a database in accordance with the present invention. In Block **600**, the recovery system **122** copies objects from the secondary data storage device to the primary data storage device. That is, each of the objects is replaced by an image copy taken at a previous time. The individual objects may be restored from the image copies concurrently with each other. In Block **602**, the recovery system **122** determines the point in the log at which to start applying operations. In Block **604**, the recovery system **122** applies log operations to all objects through one pass of the log file.

That is, beginning at the determined starting point, the recovery system **122** reads the log file and extracts the changes for each individual object. The recovery system **122** applies the changes to the specified elements within the object. The changes to an individual object may be applied concurrently with changes being applied to the other objects involved in the recovery. This continues until all the required changes have been applied to all the specified objects.

Conclusion

This concludes the description of the preferred embodiment of the invention. The following describes some alternative embodiments for accomplishing the present invention. For example, any type of computer, such as a mainframe, minicomputer, or personal computer, or computer configuration, such as a timesharing mainframe, local area network, or standalone personal computer, could be used with the present invention.

In summary, the present invention discloses a method, apparatus, and article of manufacture for a computer-implemented recovery system. The present invention provides an improved recovery system for a database. Additionally, the present invention provides recovery for partitions and partitioning indexes simultaneously. Moreover, the present invention provides a recovery system for a database that requires only one pass of a log file to apply modifications to the database.

The foregoing description of the preferred embodiment of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto.

What is claimed is:

1. A method of restoring a database in a computer, the database containing objects and being stored on a primary data storage device connected to the computer, the method comprising the steps of:

copying objects of different types in the database from the primary data storage device to a secondary data storage device, wherein one of the objects is a table index for locating data in a table, and wherein one of the objects is a partitioning index for defining a scope of each partition and thereby assigning a row of the table to its respective partition;

logging modifications to the objects, including the table index and the partitioning index, in a log file;

receiving a recovery indicator indicating that recovery of the objects in the database is required;

copying the objects, including the table index and the partitioning index, from the secondary data storage device to the database on the primary data storage device; and

applying the modifications in the log file to the copied objects, including the table index and the partitioning index, during one pass through the log file.

2. The method of claim 1, wherein the types of the objects include table data.

3. The method of claim 1, wherein the types of the objects include partition indexes.

4. The method of claim 1, wherein the recovery indicator indicates that modifications to the objects are to be reversed.

5. The method of claim 1, wherein the recovery indicator indicates that the objects have been corrupted.

6. The method of claim 1, further comprising the step of maintaining log ranges.

7. The method of claim 1, further comprising the step of maintaining copy information for objects that are copied.

8. The method of claim 7, further comprising the step of determining a starting point in the log file based on the copy information.

9. An apparatus for restoring a database in a computer, comprising:

a computer having a primary data storage device connected thereto, wherein the primary data storage device stores a database containing objects;

one or more computer programs, performed by the computer, for copying objects of different types in the database from the primary data storage device to a secondary data storage device, wherein one of the objects is a table index for locating data in a table, and wherein one of the objects is a partitioning index for defining a scope of each partition and thereby assigning a row of the table to its respective partition, logging modifications to the objects, including the table index and the partitioning index, in a log file, receiving a recovery indicator indicating that recovery of the objects in the database is required, copying the objects, including the table index and the partitioning index, from the secondary data storage device to the database on the primary data storage device, and applying the modifications in the log file to the copied objects, including the table index and the partitioning index, during one pass through the log file.

10. The apparatus of claim 9, wherein the types of the objects include table data.

11. The apparatus of claim 9, wherein the types of the objects include partition indexes.

12. The apparatus of claim 9, wherein the recovery indicator indicates that modifications to the objects are to be reversed.

13. The apparatus of claim 9, wherein the recovery indicator indicates that the objects have been corrupted.

14. The apparatus of claim 9, further comprising the means for maintaining log ranges.

15. The apparatus of claim 9, further comprising the means for maintaining copy information for objects that are copied.

16. The apparatus of claim 15, further comprising the means for determining a starting point in the log file based on the copy information.

17. An article of manufacture comprising a computer program carrier readable by a computer and embodying one or more instructions executable by the computer to perform method steps for restoring a database, the database containing objects and being stored on a primary data storage device connected to the computer, the method comprising the steps of:

copying objects of different types in the database from the primary data storage device to a secondary data storage device, wherein one of the objects is a table index for locating data in a table, and wherein one of the objects is a partitioning index for defining a scope of each partition and thereby assigning a row of the table to its respective partition;

logging modifications to the objects, including the table index and the partitioning index, in a log file;

receiving a recovery indicator indicating that recovery of the objects in the database is required;

copying the objects, including the table index and the partitioning index, from the secondary data storage device to the database on the primary data storage device; and

applying the modifications in the log file to the copied objects, including the table index and the partitioning index, during one pass through the log file.

18. The method of claim 17, wherein the types of the objects include table data.

19. The method of claim 17, wherein the types of the objects include partition indexes.

20. The method of claim 17, wherein the recovery indicator indicates that modifications to the objects are to be reversed.

21. The method of claim 17, wherein the recovery indicator indicates that the objects have been corrupted.

22. The method of claim 17, further comprising the step of maintaining log ranges.

23. The method of claim 17, further comprising the step of maintaining copy information for objects that are copied.

24. The method of claim 23, further comprising the step of determining a starting point in the log file based on the copy information.

* * * * *

Printed by EAST

UserID: CNguyen14
Computer: WS04871
Date: 10/19/2002
Time: 08:31

Document Listing

Document	Image pages	Text pages	Error pages
US 6178425 B1	28	0	0
Total	28	0	0

US00RE37857E

(19) **United States**

(12) **Reissued Patent**

Browne

(10) **Patent Number: US RE37,857 E**

(45) **Date of Reissued Patent: Sep. 24, 2002**

(54) **DATA PROCESSING SYSTEM FOR COMMUNICATIONS NETWORK**

(75) Inventor: **John Martin Browne**, Surrey (GB)

(73) Assignee: **British Telecommunications public limited company**, London (GB)

(*) Notice: This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/797,773**

(22) PCT Filed: **Mar. 31, 1994**

(86) PCT No.: **PCT/GB94/00706**

§ 371 (c)(1),
(2), (4) Date: **Mar. 6, 1995**

(87) PCT Pub. No.: **WO94/23530**

PCT Pub. Date: **Oct. 13, 1994**

Related U.S. Patent Documents

Reissue of:

(64) Patent No.: **5,768,353**
Issued: **Jun. 16, 1998**
Appl. No.: **08/392,976**
Filed: **Mar. 6, 1995**

(30) **Foreign Application Priority Data**

Mar. 31, 1993 (GB) 9306724
Mar. 31, 1993 (GB) 9306725
Aug. 24, 1993 (GB) 9317619

(51) **Int. Cl.**[7] **H04M 3/22**; H04M 3/08; H04M 15/00

(52) **U.S. Cl.** **379/114.1**; 379/111; 379/13; 379/28

(58) **Field of Search** 379/103, 13, 27, 379/28, 34, 111, 112.05, 113, 114.03, 115.01, 207.02, 219, 220.01, 221.02

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,813,495 A 5/1974 Conerly

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

CA	2101305	1/1997
EP	452 591	10/1991
EP	311 607	12/1991
EP	462 726	12/1991
EP	477628	4/1992
EP	483 091	4/1992
EP	483 857	5/1992
EP	491 497	6/1992
EP	569 175	11/1993
GB	2 246 051	1/1992
GB	2254224	9/1992
GB	2 255 690	11/1992
WO	93/08661	4/1993

OTHER PUBLICATIONS

L. Swann, "Universal Operations Systems—Integrated Building Blocks", AT&T Technology, vol. 1, No. 1, 1986, pp. 66–71.

(List continued on next page.)

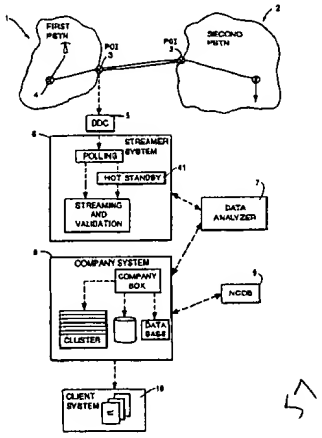
Primary Examiner—Vijay Shankar

(74) *Attorney, Agent, or Firm*—Nixon & Vanderhye PC

(57) **ABSTRACT**

An inter-network call accounting system for use in a communication network such as the public switched telephone network in Britain allows call records to be sorted according to the network operator to be charged in respect of the calls, prior to being priced and charged. A data analyzer incorporating an expert system is provided for call records which cannot be validated. The data analyzer can apply default or amended data, or can output invalid data to a suspended process awaiting updated reference information. Unfixable data is output to a sump for management purposes. A pricing and charging engine processes data already sorted according to billable entity and incorporates further data analysis for dealing with data invalid by reason of pricing and charging related information.

29 Claims, 35 Drawing Sheets



U.S. PATENT DOCUMENTS

4,727,577	A	2/1988	Frey et al.
4,813,065	A	3/1989	Segala
4,839,916	A	6/1989	Fields
4,979,207	A	12/1990	Baum et al. 379/112
5,003,584	A	3/1991	Benyacar et al.
5,008,929	A	4/1991	Olsen
5,042,027	A	8/1991	Takase et al.
5,063,591	A	11/1991	Jodoin
5,103,475	A	4/1992	Shuen
5,185,785	A	2/1993	Funk et al.
5,218,632	A	6/1993	Cool 379/126
5,265,155	A	11/1993	Castro
5,274,695	A	12/1993	Green
5,333,183	A	7/1994	Herbert
5,335,268	A	8/1994	Kelly
5,355,403	A	10/1994	Richardson et al.
5,369,680	A	11/1994	Borbas
5,488,648	A	1/1996	Womble
5,517,560	A	5/1996	Greenspan

OTHER PUBLICATIONS

A.C. Stark, Jr. et al., "Harnessing Intelligent Networks", AT&T Technology, vol. 2, No. 3, 1987, pp. 14–21.

L.E. Murphy et al., "Managing Corporate Data Networks", AT&T Technology, vol. 2, No. 3, 1987, pp. 40–47.

J.G. Brinsfield et al., "Unified Network Management Architecture—Putting It All Together", AT&T Technology, vol. 3, No. 2, 1988, pp. 6–17.

D.J. Desmond, "Centralized Management of VCS Networks", AT&T Technology, vol. 3, No. 2, 1988, pp. 34–41.

A.L. Barrese, "Open Systems and AT&T's Application Operating Environment", AT&T Technology, vol. 3, No. 3, 1988, pp. 6–15.

R.H. Goldberg et al., "FTS2000–Blueprint For A State–of–The–Art Digital Network", AT&T Technology, vol. 4, No. 4, 1989, pp. 6–11.

L.M. Falletta et al., "Using The Network For ISDN Support", AT&T Technology, vol. 5, No. 1, 1990, pp. 45–47.

W.F. Daniel et al., "Quality of Maintenance Advanced by PBXpert", AT&T Technology, vol. 5, No. 3, 1990, pp. 16–21.

J.K. Doyle et al., "The StarServe FT Computer Satisfies Customer Fault Tolerant Computer Needs", AT&T Technology, vol. 5, No. 4, 1990, pp. 30–35.

M. Hoover, "Managing Complex Data Networks: Start Smart", AT&T Technology, vol. 6, No. 2, 1991, pp. 38–45.

M.H. Mortensen, "Special Services Automation", AT&T Technology, vol. 4, No. 1, 1989, pp. 44–49.

S.L. Kay et al., "Data Networking With The DATAKIT II Virtual Circuit Switch", AT&T Technology, vol. 4, No. 3, 1989, pp. 20–27.

M.J. Brown, "PRX/A—A Living System", Trends in Telecommunications, vol. 3, No. 1, 1987, pp. 25–32, 34.

"PRX/A, A Living System", Trends in Telecommunications, vol. 3, No. 2, 1987, p. 78.

"MFOS: An Open Architecture for Operations Systems", Trends in Telecommunications, vol. 3, No. 2, 1987, pp. 88–89.

H. Bean et al., "5ESS–PRX as a Gateway Switch", Trends in Telecommunications, vol. 4, No. 2, 1988, pp. 53–58.

P.M. Holland et al., "From Concept to Realisation", Trends in Telecommunications, vol. 5, No 1, 1989, pp. 15–22.

M.G. Andrews et al., "Network Traffic Management and its Application for Intelligent Networks", Trends in Telecommunications, Jun. 1990, vol. 6, No. 1, pp. 13–22.

H.A. Bauer et al., "Evolution of Intelligence in Switched Networks", Trends in Telecommunications, vol. 6, No. 1, 1990, pp. 23–31.

R.C. Ward et al., "The DDSN: On Line in the United Kingdom", Trends in Telecommunications, vol. 6, No. 1, 1990, pp. 33–44.

A.P.M. Koenraadt, "EMS–2000 Element Management System", Trends in Telecommunications, vol. 7, No. 2, 1991, pp. 68–71.

S. Abujawdeh et al., "Operation Systems—The Heart of Providing Quality Services", Trends in Telecommunications, vol. 7, No. 3, 1991, pp. 44–51.

Wes Czajkowski et al., "5ESS Switch Bringing ETSI–ISDN to the Market", Trends in Telecommunications, vol. 8, No. 2, 1992, pp. 23–38.

T. Adelman et al., "Flexcab Release 2.0 System Blueprint Final Version 1.0", Dec. 1992, pp. I.1–VII.1, 1–18.

"BOC Notes on the LEC Networks—1990", Issue Mar. 1, 1991.

"Framework Generic Requirements for the Second Generation of Automatic Message Accounting Data Networking System", Issue 1, Sep. 1992.

"Generic Requirements for the Second Generation of Automatic Message Accounting Teleprocessing Systems", Issue 1, Apr. 1991.

"Automatic Message Accounting Teleprocessing System (AMATPS) Collector Interface Requirements", Issue 1, Nov. 1984.

J. McDermott, "Telephone Traffic Management in OTC(A)", Network Management And Teletraffic Engineering Seminar, Sydney Australia, Feb. 26–Mar. 7, 1985.

J. McDermott, "Service Performance Analysis of the OTC(A) Telex Network", Network Management And Teletraffic Engineering Seminar, Sydney, Australia, Mar. 1985.

R.O.G. Jacobs, "The Centoc Daily Traffic Recording System", Telecom Journal of Australia, vol. 33, No. 3, 1983, pp. 235–242.

J. Berrier et al., "International Cooperation results in Successful Installation", Telephony, vol. 208, No. 25, 1985, pp. 32–40.

"BT's Cellnet Retailers Get Advanced Billing System", DEC User, Oct. 1985, p. 25.

R. Ahmari et al., "Helping Switches to Account for Themselves", Record, AT&T Bell Labs, vol. 63, No. 2, 1985, pp. 16–22.

J. Robinson, Jr., "Versatile System Takes the Terror out of Billing", Telephone Engineer & Management, vol. 87, No. 7, 1983, pp. 63–65.

"Request for Information (RFI) on Automatic Message Account (AMA) Data Networking System", Bellcore, Issue 1, Apr. 1991.

M. Bhattacharya "Computerised Telephone Revenue Billing, Accounting and Control System", Telecommunications, Dec. 1989, pp. 45–50.

P. Webster, "Practical Data Applications of Cellular Networks", Second National Conference on Cellular Communications, Conference Papers, Feb. 1986, pp. 193–224.
Shimizu et al., "New Automatic Message Accounting System", Japan Telecommunication Review, vol. 28, 1986, pp. 163–172.
FLEXCAB Flexible Charging and Billing System, pp. 1–182 (Undated).
"AOTC Functional Design", 027.001–RDCSAUSDS–P1.0, pp. 1–13–6, A–1–A–6 (Undated).
E.J. Obuchowski, "Access Charge and Revenue Architecture", AT&T Technical Journal, vol. 66, No. 3, May 1987, pp. 73–81.
B. Grynko, "Mise en place d'un réseau'd exploitation pour la communitation électronique", Jun. 1987, vol. 26, No. 2, pp. 18–23 (Translation Also Enclosed).
Savoy et al., "The Rata System", Mar. 1979, pp. 78–89 (Translation Also Enclosed).

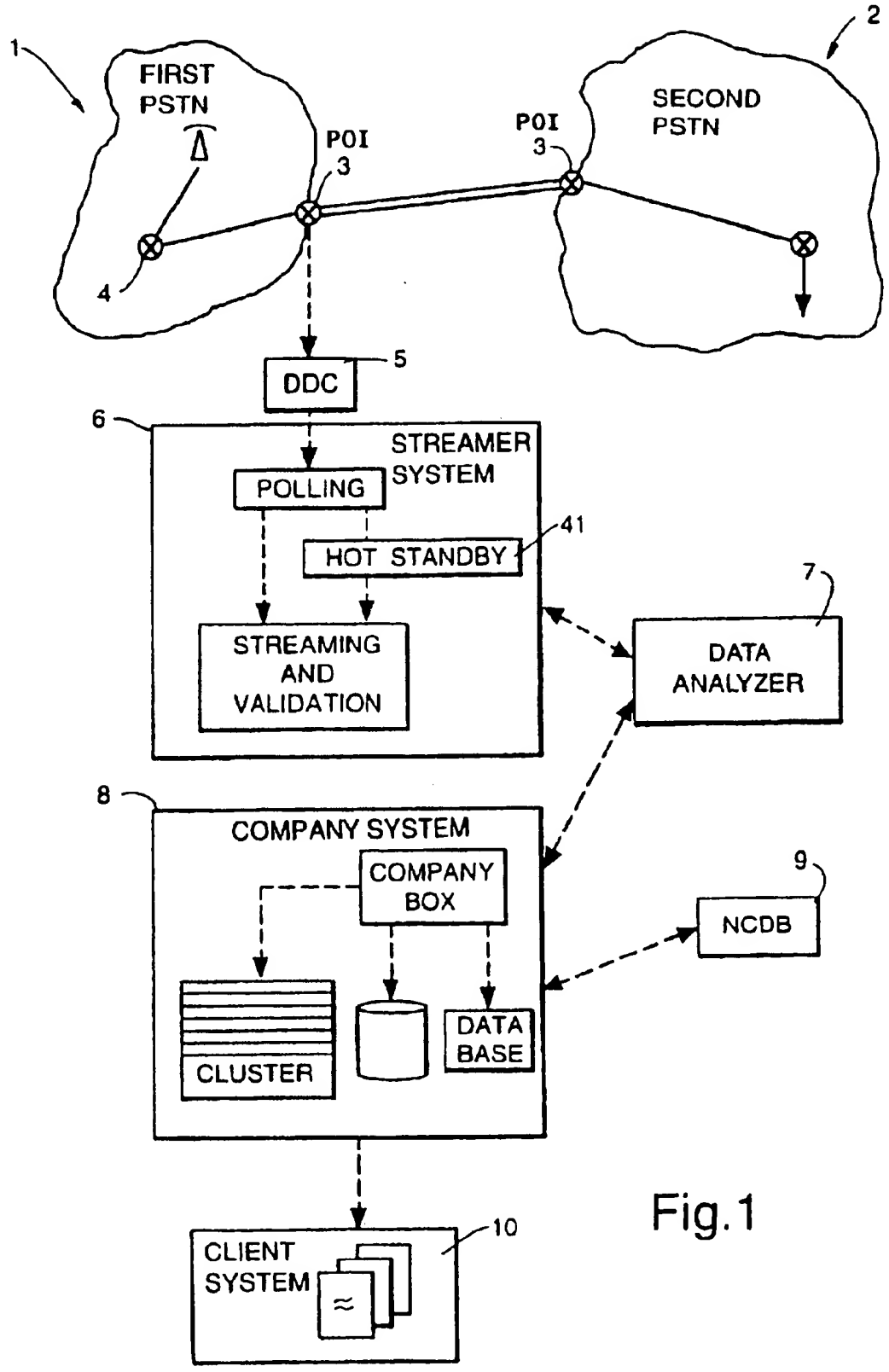

1
FIRST PSTN
POI 3
4
2
SECOND PSTN
POI 3
DDC
5
6
STREAMER SYSTEM
POLLING
HOT STANDBY
41
STREAMING AND VALIDATION
7
DATA ANALYZER
8
COMPANY SYSTEM
COMPANY BOX
9
NCDB
CLUSTER
DATA BASE
CLIENT SYSTEM
10

Fig.1

Fig.2

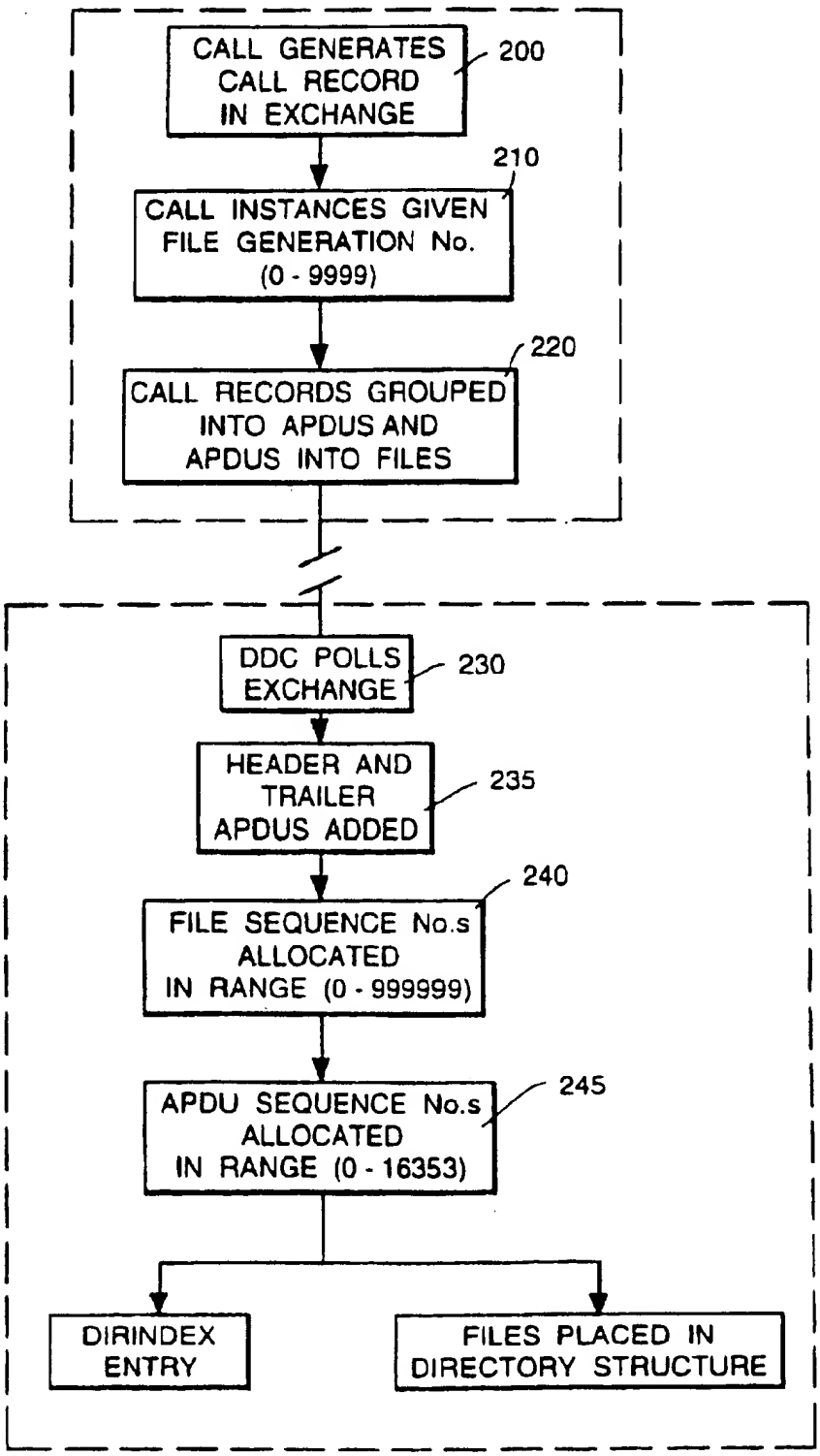
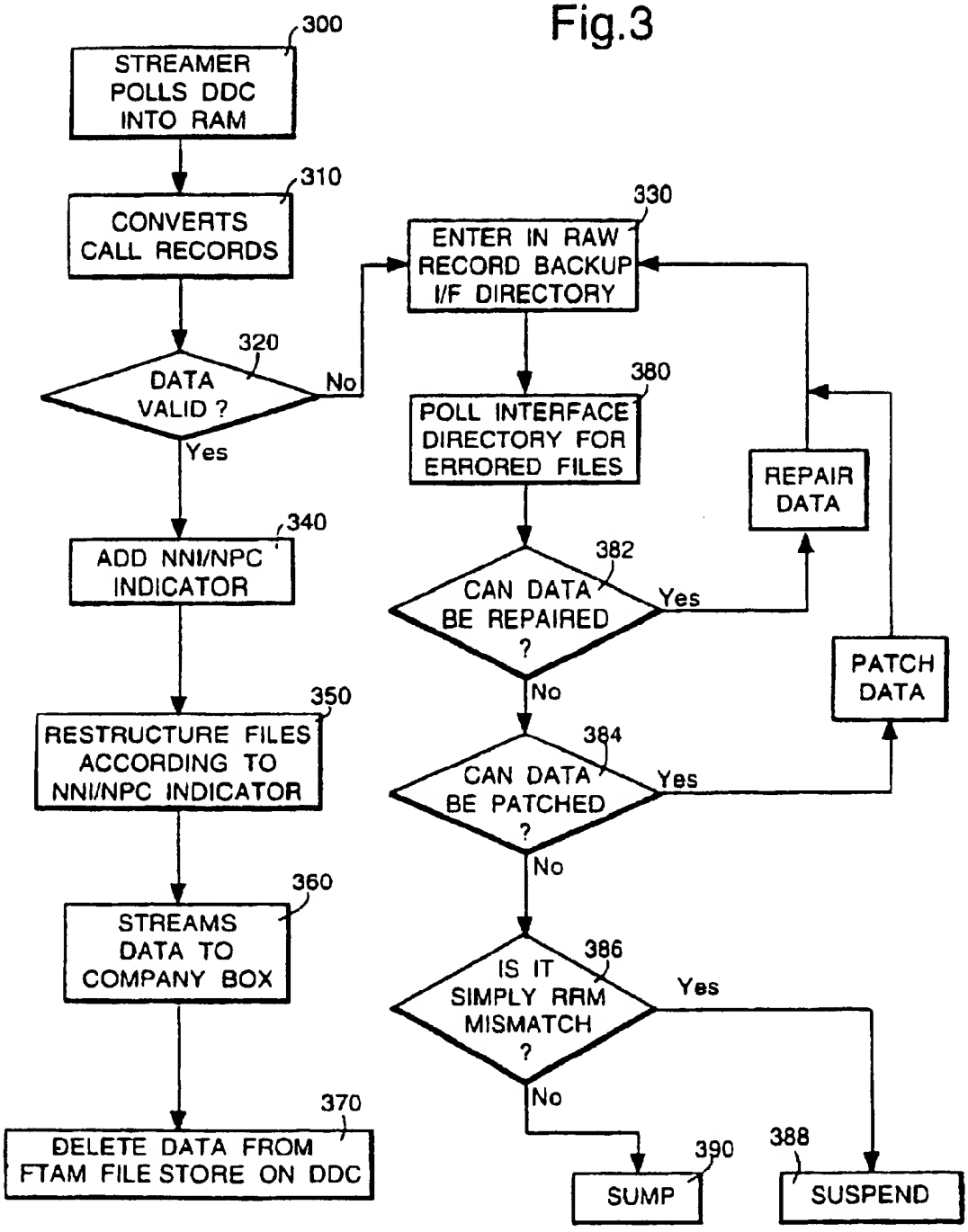

CALL GENERATES CALL RECORD IN EXCHANGE
200
CALL INSTANCES GIVEN FILE GENERATION No. (0 - 9999)
210
CALL RECORDS GROUPED INTO APDUS AND APDUS INTO FILES
220
DDC POLLS EXCHANGE
230
HEADER AND TRAILER APDUS ADDED
235
FILE SEQUENCE No.s ALLOCATED IN RANGE (0 - 999999)
240
APDU SEQUENCE No.s ALLOCATED IN RANGE (0 - 16353)
245
DIRINDEX ENTRY
FILES PLACED IN DIRECTORY STRUCTURE

Fig.3


300
STREAMER POLLS DDC INTO RAM
310
CONVERTS CALL RECORDS
320
DATA VALID ?
No
Yes
340
ADD NNI/NPC INDICATOR
350
RESTRUCTURE FILES ACCORDING TO NNI/NPC INDICATOR
360
STREAMS DATA TO COMPANY BOX
370
DELETE DATA FROM FTAM FILE STORE ON DDC
330
ENTER IN RAW RECORD BACKUP I/F DIRECTORY
380
POLL INTERFACE DIRECTORY FOR ERRORED FILES
382
CAN DATA BE REPAIRED ?
Yes
No
REPAIR DATA
384
CAN DATA BE PATCHED ?
Yes
No
PATCH DATA
386
IS IT SIMPLY RRM MISMATCH ?
Yes
No
390
SUMP
388
SUSPEND

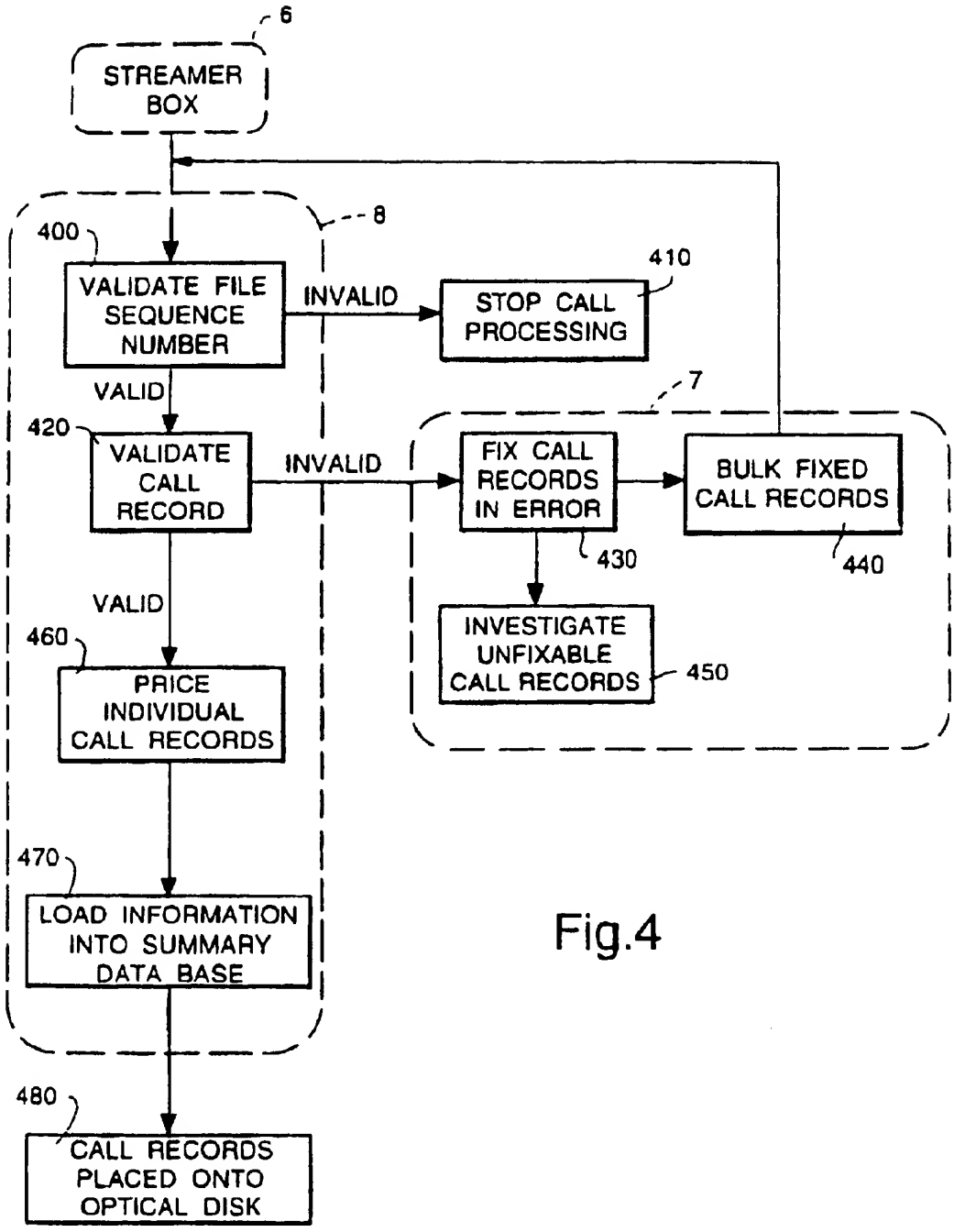

6
STREAMER BOX
8
400
VALIDATE FILE SEQUENCE NUMBER
INVALID
410
STOP CALL PROCESSING
VALID
7
420
VALIDATE CALL RECORD
INVALID
FIX CALL RECORDS IN ERROR
BULK FIXED CALL RECORDS
430
440
VALID
460
PRICE INDIVIDUAL CALL RECORDS
INVESTIGATE UNFIXABLE CALL RECORDS
450
470
LOAD INFORMATION INTO SUMMARY DATA BASE
480
CALL RECORDS PLACED ONTO OPTICAL DISK

Fig.4

Fig.5

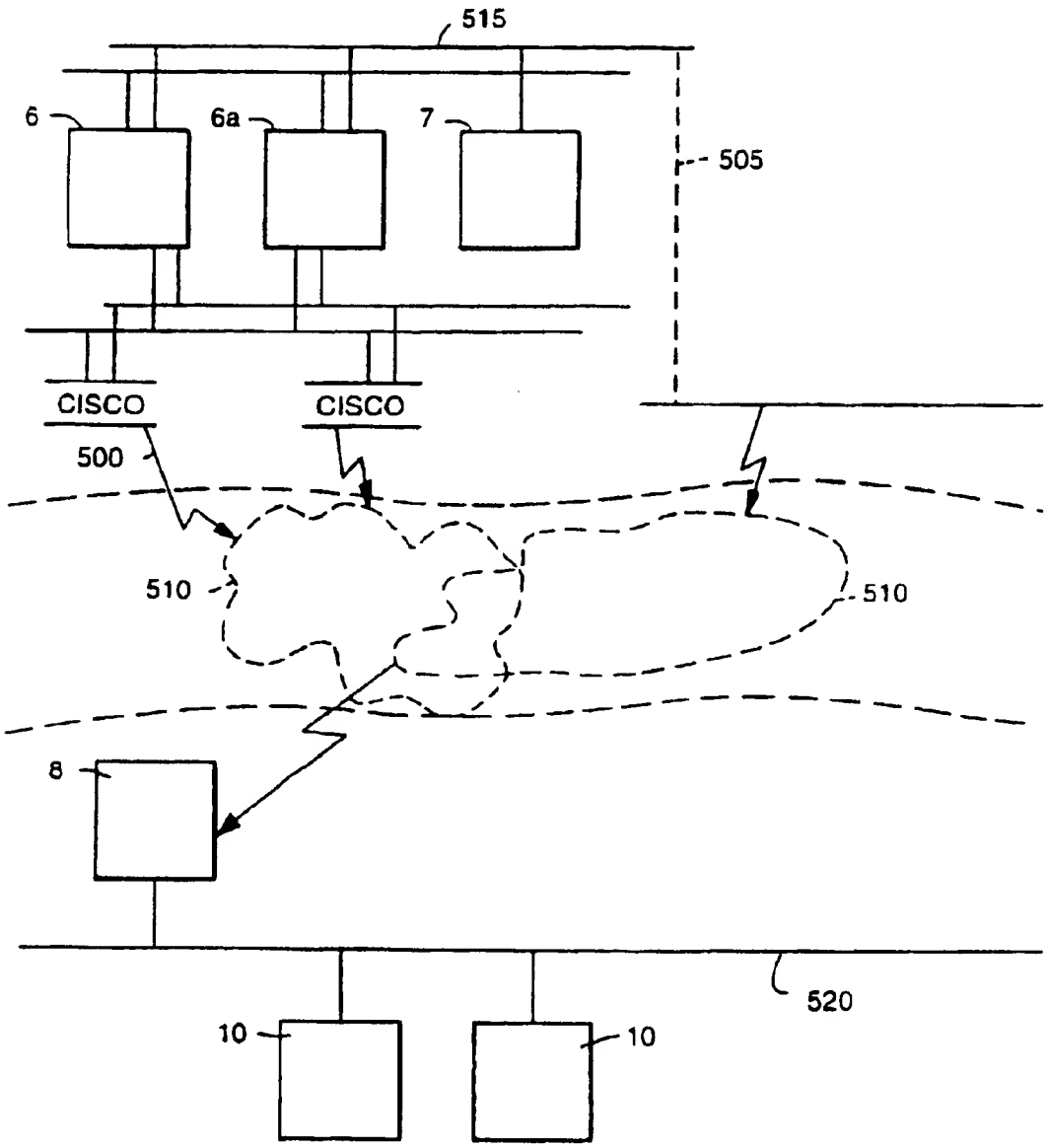


Fig.6

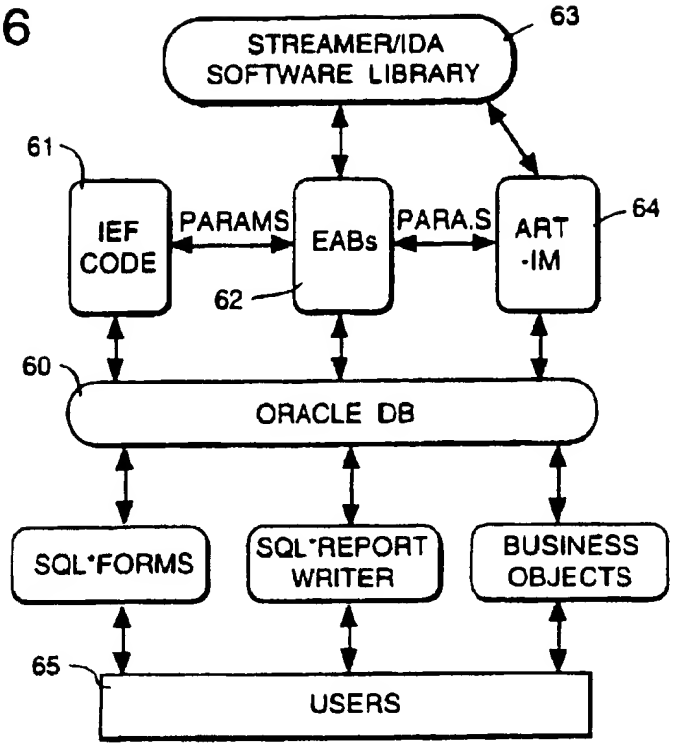

STREAMER/IDA SOFTWARE LIBRARY
63
61
IEF CODE
PARAMS
EABs
PARA.S
ART -IM
64
62
60
ORACLE DB
SQL*FORMS
SQL*REPORT WRITER
BUSINESS OBJECTS
65
USERS

Fig.7

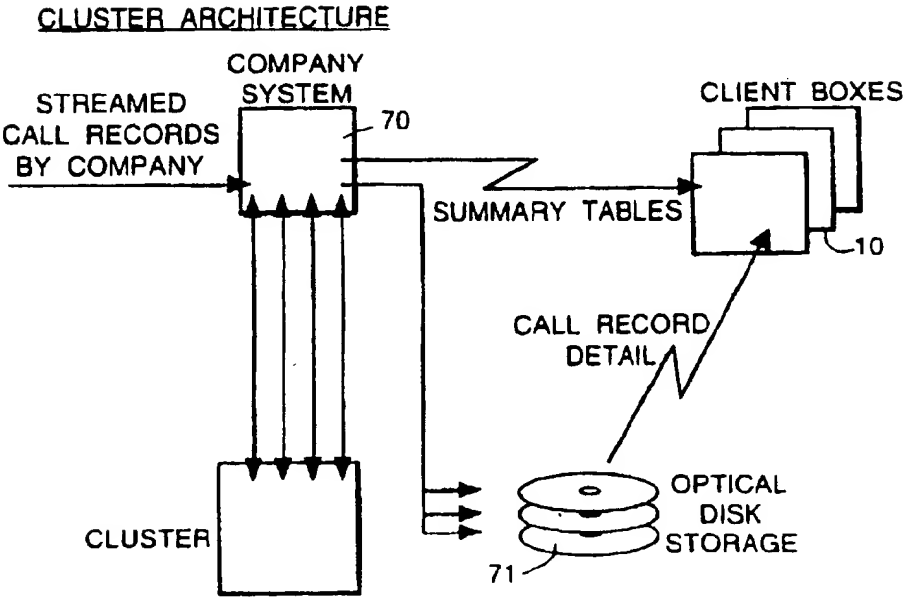
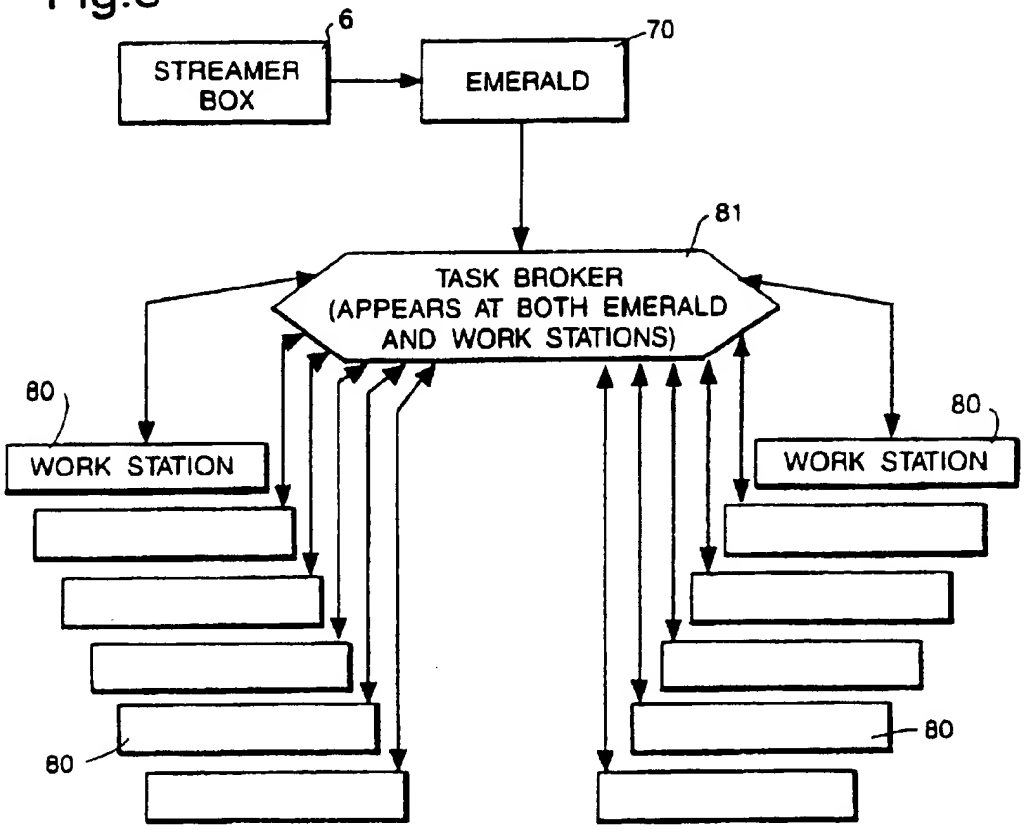

CLUSTER ARCHITECTURE
COMPANY SYSTEM
STREAMED CALL RECORDS BY COMPANY
70
CLIENT BOXES
SUMMARY TABLES
10
CALL RECORD DETAIL
OPTICAL DISK STORAGE
71
CLUSTER

Fig.8


6
STREAMER BOX
70
EMERALD
81
TASK BROKER
(APPEARS AT BOTH EMERALD AND WORK STATIONS)
80
WORK STATION
80
WORK STATION
80
80

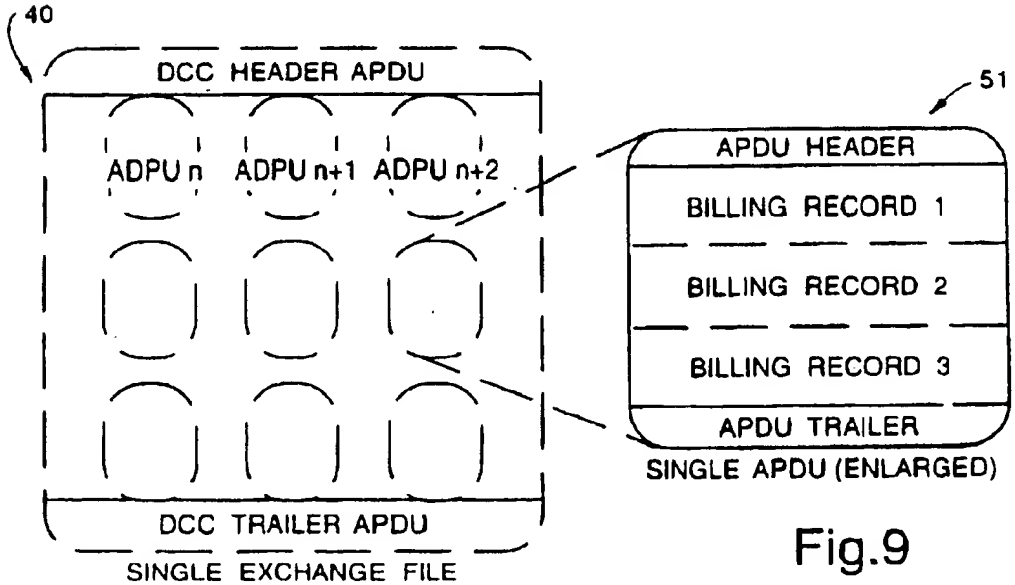

40
DCC HEADER APDU
ADPU n
ADPU n+1
ADPU n+2
DCC TRAILER APDU
SINGLE EXCHANGE FILE
51
APDU HEADER
BILLING RECORD 1
BILLING RECORD 2
BILLING RECORD 3
APDU TRAILER
SINGLE APDU (ENLARGED)

Fig.9

Fig.10

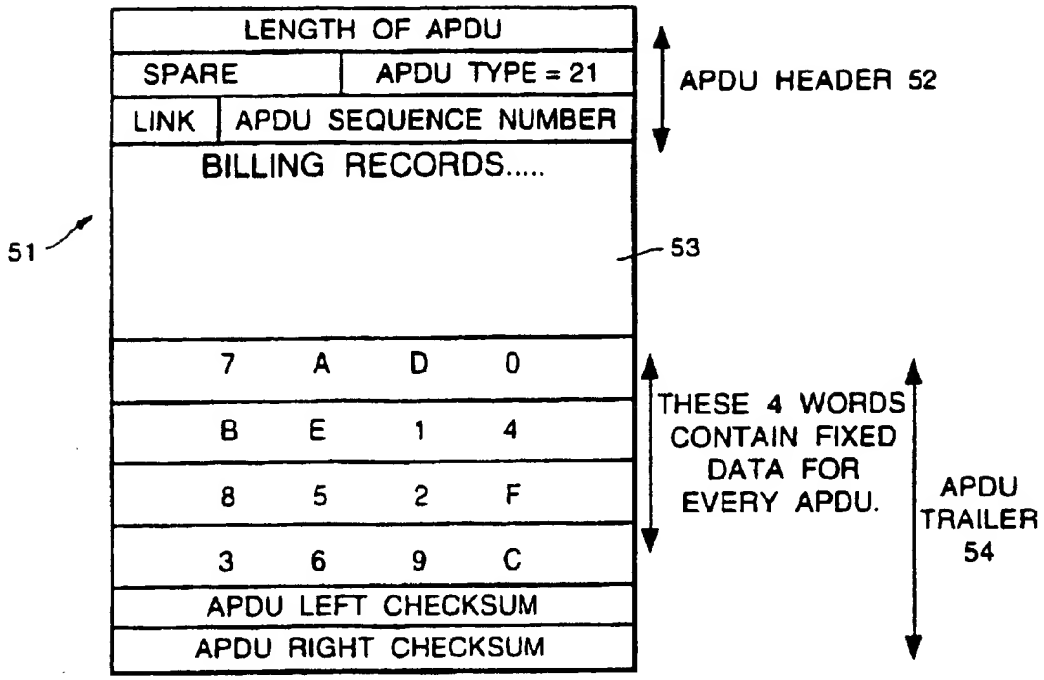
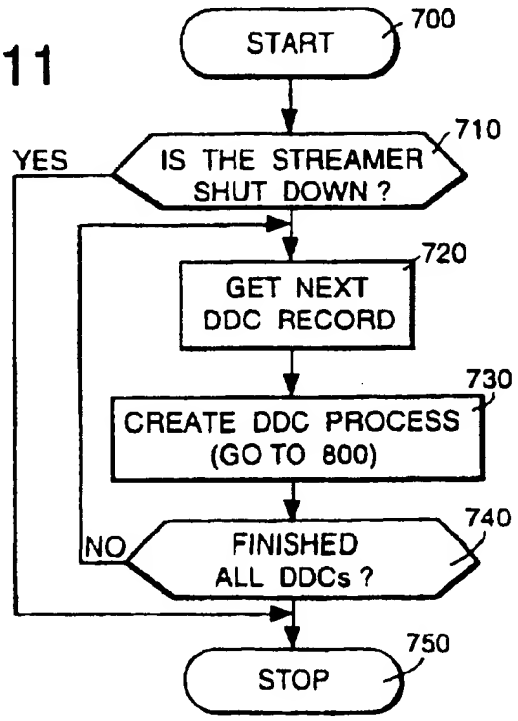

LENGTH OF APDU
SPARE
APDU TYPE = 21
LINK
APDU SEQUENCE NUMBER
BILLING RECORDS.....
51
53
7 A D 0
B E 1 4
8 5 2 F
3 6 9 C
APDU LEFT CHECKSUM
APDU RIGHT CHECKSUM
APDU HEADER 52
THESE 4 WORDS CONTAIN FIXED DATA FOR EVERY APDU.
APDU TRAILER 54

Fig.11


START
700
IS THE STREAMER SHUT DOWN ?
710
YES
GET NEXT DDC RECORD
720
CREATE DDC PROCESS (GO TO 800)
730
FINISHED ALL DDCs ?
740
NO
STOP
750

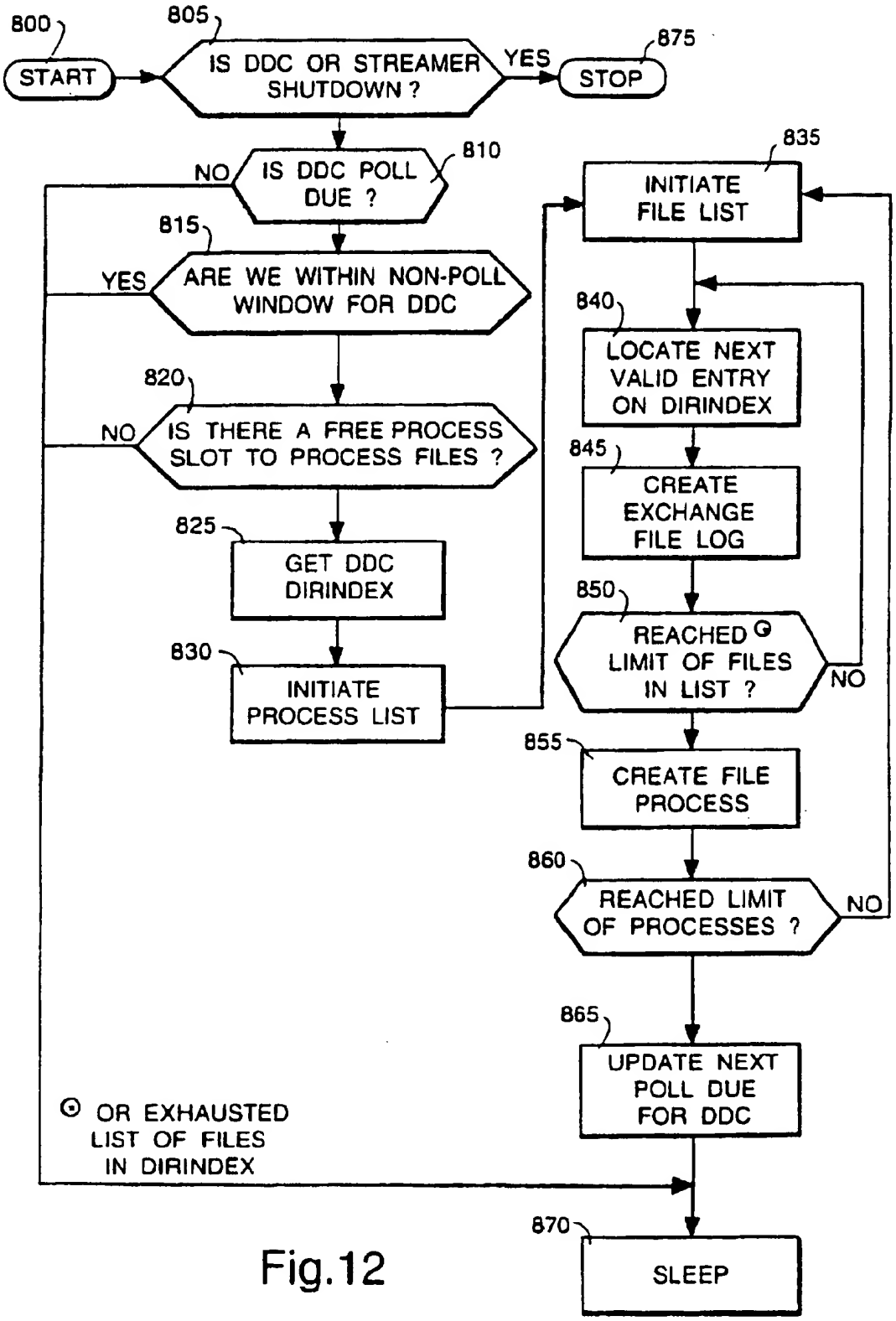

800
START
805
IS DDC OR STREAMER SHUTDOWN ?
YES
875
STOP
NO
810
IS DDC POLL DUE ?
815
YES
ARE WE WITHIN NON-POLL WINDOW FOR DDC
820
NO
IS THERE A FREE PROCESS SLOT TO PROCESS FILES ?
825
GET DDC DIRINDEX
830
INITIATE PROCESS LIST
835
INITIATE FILE LIST
840
LOCATE NEXT VALID ENTRY ON DIRINDEX
845
CREATE EXCHANGE FILE LOG
850
REACHED ⊙ LIMIT OF FILES IN LIST ?
NO
855
CREATE FILE PROCESS
860
REACHED LIMIT OF PROCESSES ?
NO
865
UPDATE NEXT POLL DUE FOR DDC
⊙ OR EXHAUSTED LIST OF FILES IN DIRINDEX
870
SLEEP

Fig.12

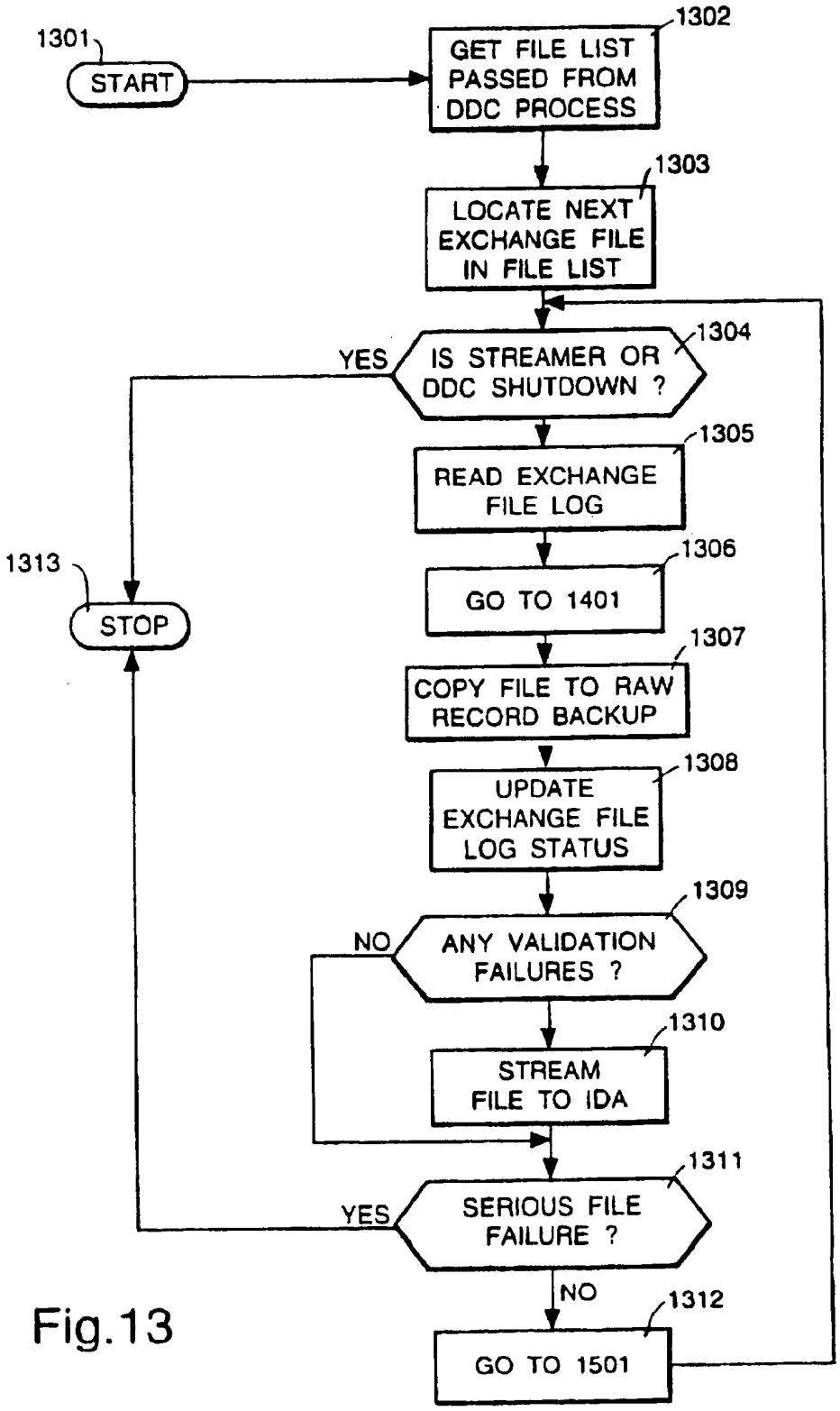

1301
START
1302
GET FILE LIST PASSED FROM DDC PROCESS
1303
LOCATE NEXT EXCHANGE FILE IN FILE LIST
1304
YES
IS STREAMER OR DDC SHUTDOWN ?
1305
READ EXCHANGE FILE LOG
1306
GO TO 1401
1307
COPY FILE TO RAW RECORD BACKUP
1308
UPDATE EXCHANGE FILE LOG STATUS
1309
NO
ANY VALIDATION FAILURES ?
1310
STREAM FILE TO IDA
1311
YES
SERIOUS FILE FAILURE ?
NO
1312
GO TO 1501
1313
STOP

Fig.13

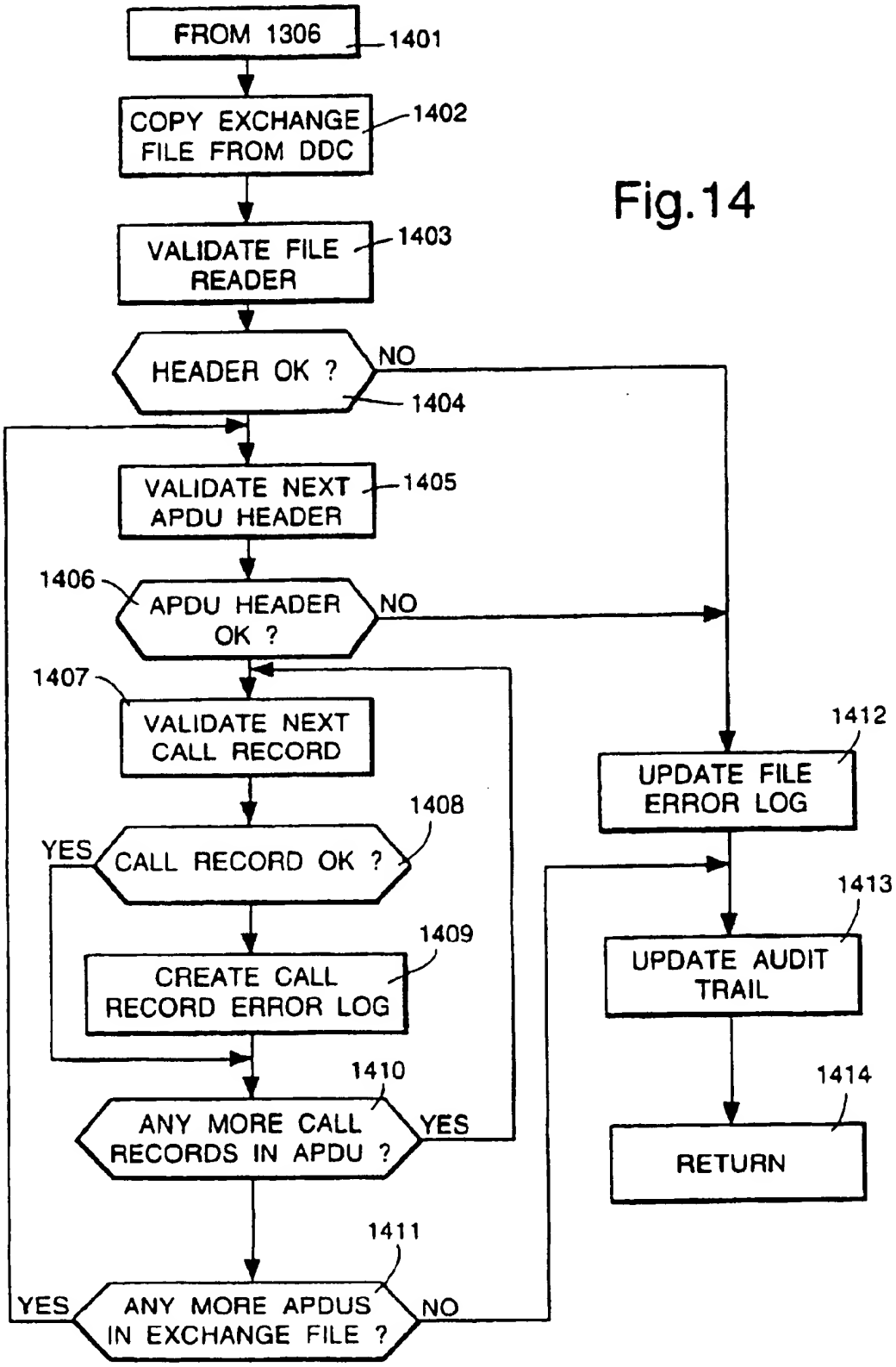

FROM 1306
1401
COPY EXCHANGE FILE FROM DDC
1402
VALIDATE FILE READER
1403
HEADER OK ?
NO
1404
VALIDATE NEXT APDU HEADER
1405
1406
APDU HEADER OK ?
NO
1407
VALIDATE NEXT CALL RECORD
1408
YES
CALL RECORD OK ?
1409
CREATE CALL RECORD ERROR LOG
1410
ANY MORE CALL RECORDS IN APDU ?
YES
1411
YES
ANY MORE APDUS IN EXCHANGE FILE ?
NO
1412
UPDATE FILE ERROR LOG
1413
UPDATE AUDIT TRAIL
1414
RETURN

Fig.14

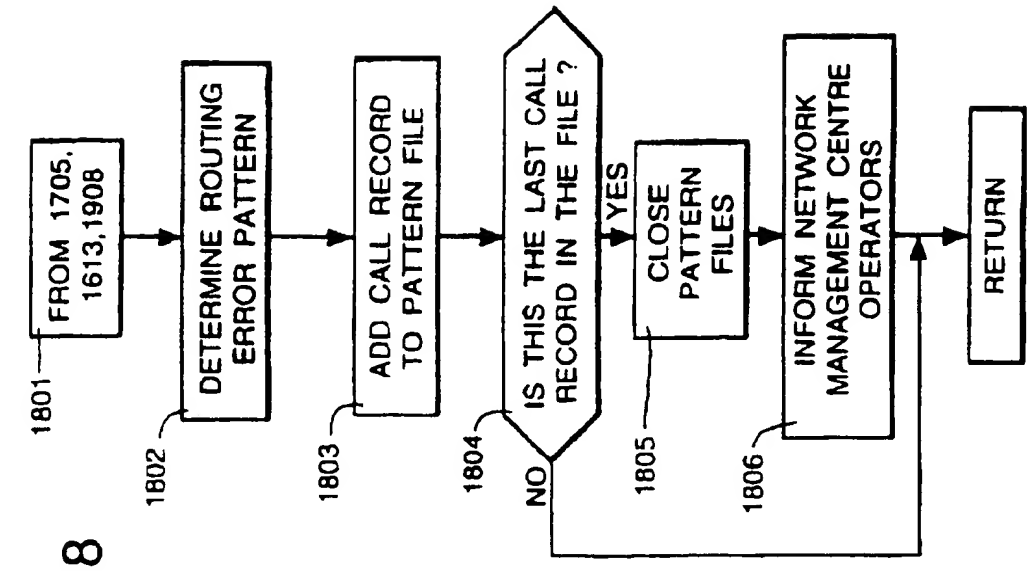

1801
FROM 1705, 1613, 1908
1802
DETERMINE ROUTING ERROR PATTERN
1803
ADD CALL RECORD TO PATTERN FILE
1804
IS THIS THE LAST CALL RECORD IN THE FILE ?
YES
NO
1805
CLOSE PATTERN FILES
1806
INFORM NETWORK MANAGEMENT CENTRE OPERATORS
RETURN

Fig.18

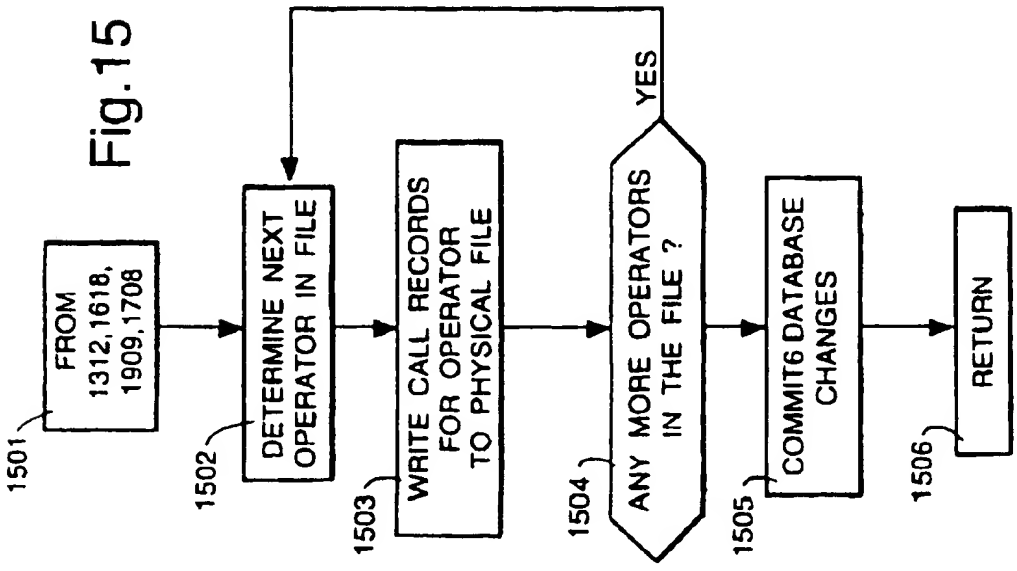

1501
FROM 1312, 1618, 1909, 1708
1502
DETERMINE NEXT OPERATOR IN FILE
1503
WRITE CALL RECORDS FOR OPERATOR TO PHYSICAL FILE
1504
ANY MORE OPERATORS IN THE FILE ?
YES
1505
COMMIT6 DATABASE CHANGES
1506
RETURN

Fig.15

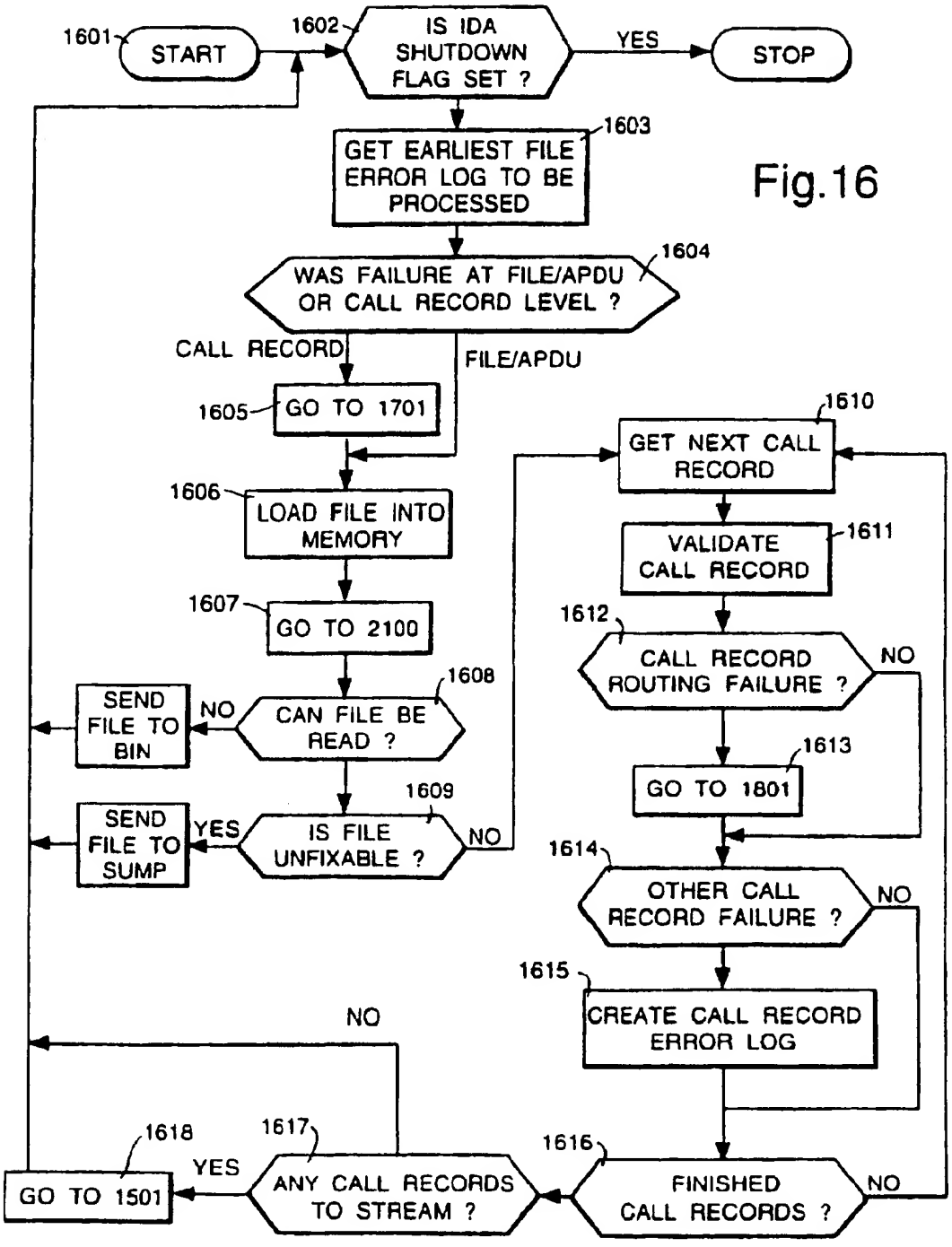

1601
START
1602
IS IDA SHUTDOWN FLAG SET ?
YES
STOP
1603
GET EARLIEST FILE ERROR LOG TO BE PROCESSED
1604
WAS FAILURE AT FILE/APDU OR CALL RECORD LEVEL ?
CALL RECORD
FILE/APDU
1605
GO TO 1701
1606
LOAD FILE INTO MEMORY
1607
GO TO 2100
1608
CAN FILE BE READ ?
NO
SEND FILE TO BIN
1609
IS FILE UNFIXABLE ?
YES
SEND FILE TO SUMP
NO
1610
GET NEXT CALL RECORD
1611
VALIDATE CALL RECORD
1612
CALL RECORD ROUTING FAILURE ?
NO
1613
GO TO 1801
1614
OTHER CALL RECORD FAILURE ?
NO
1615
CREATE CALL RECORD ERROR LOG
1616
FINISHED CALL RECORDS ?
NO
1617
ANY CALL RECORDS TO STREAM ?
YES
NO
1618
GO TO 1501

Fig.16

Fig.17

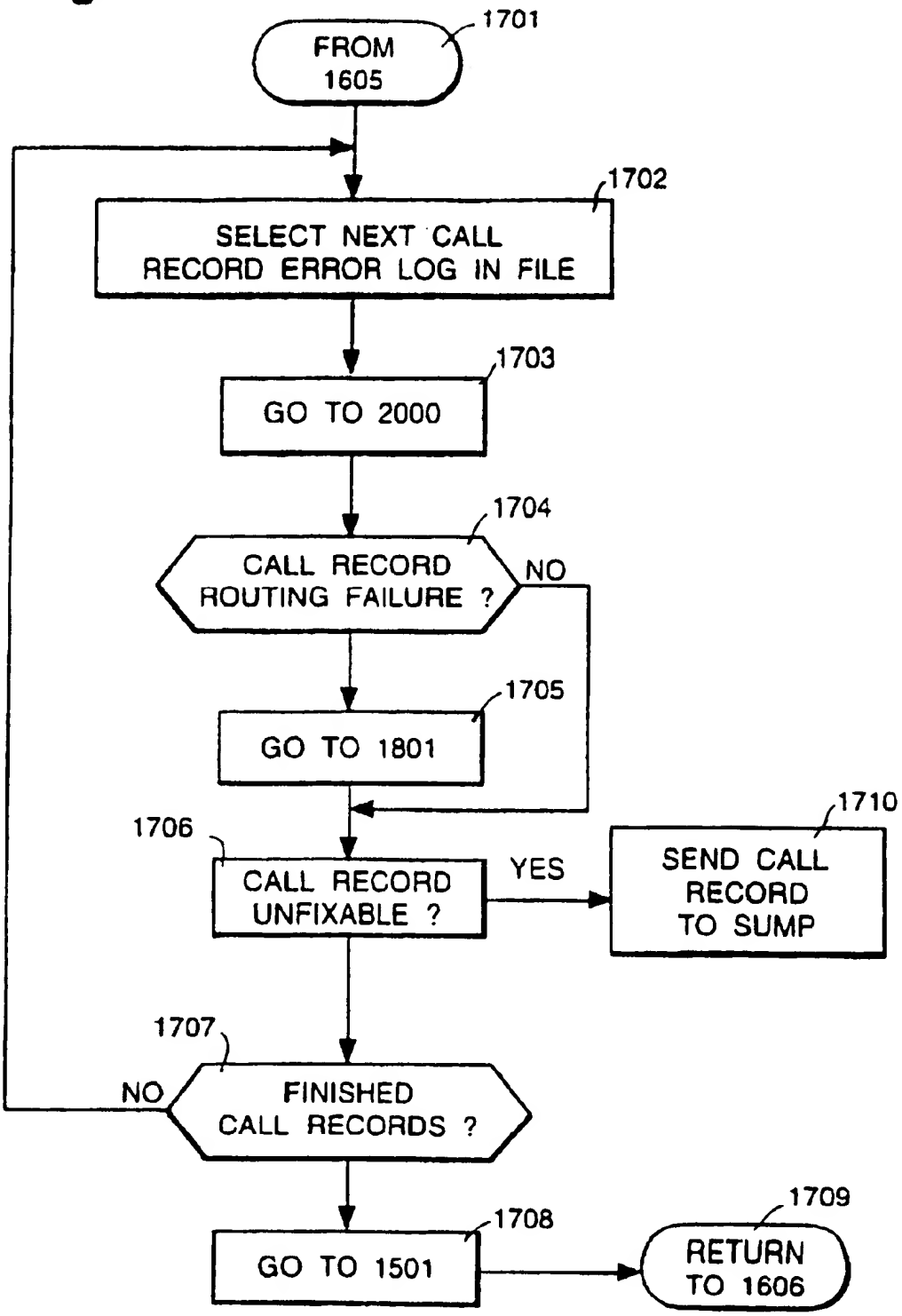

1701
FROM 1605
1702
SELECT NEXT CALL RECORD ERROR LOG IN FILE
1703
GO TO 2000
1704
CALL RECORD ROUTING FAILURE ?
NO
1705
GO TO 1801
1706
CALL RECORD UNFIXABLE ?
YES
1710
SEND CALL RECORD TO SUMP
1707
NO
FINISHED CALL RECORDS ?
1708
GO TO 1501
1709
RETURN TO 1606

Fig.19

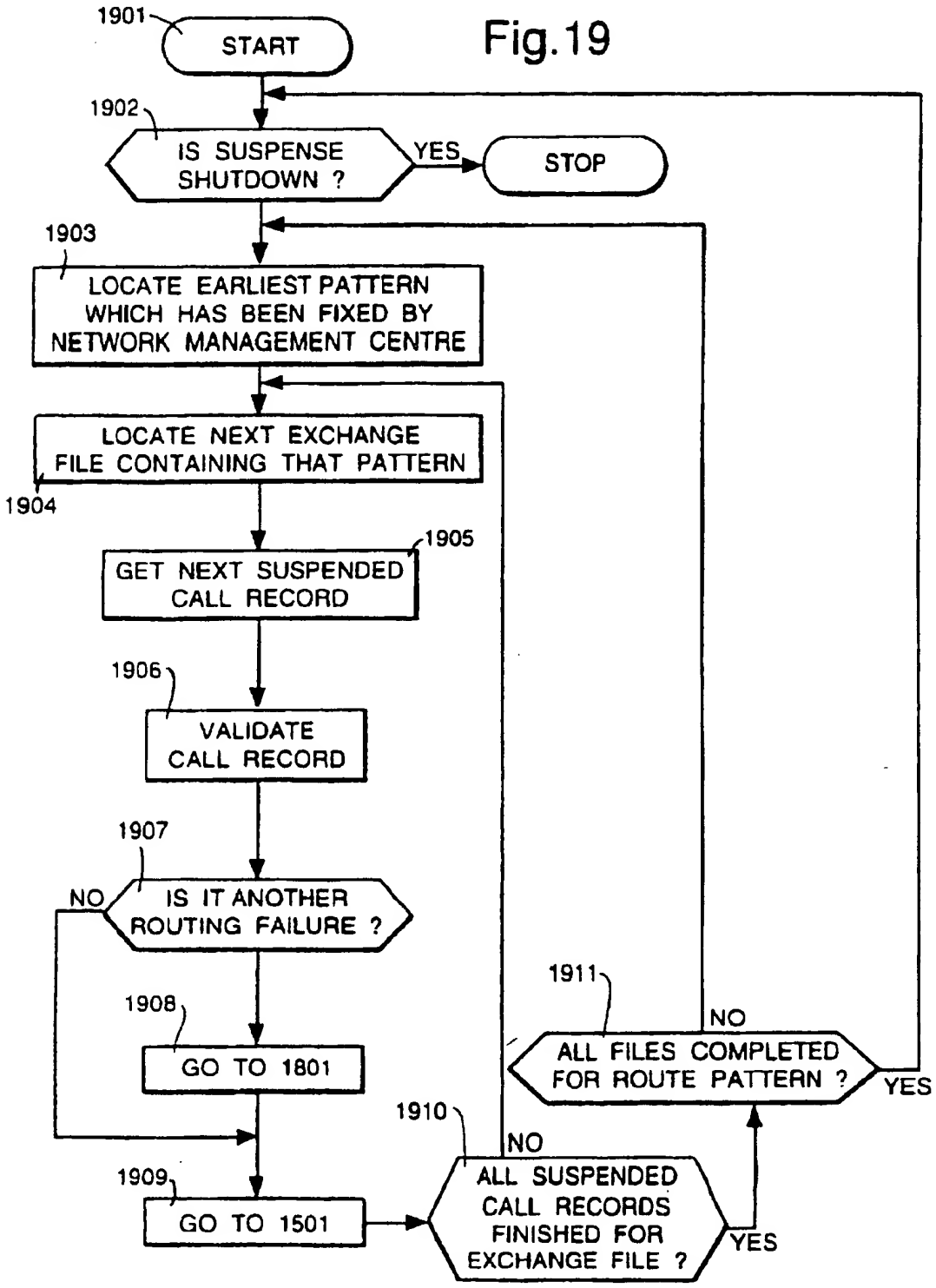
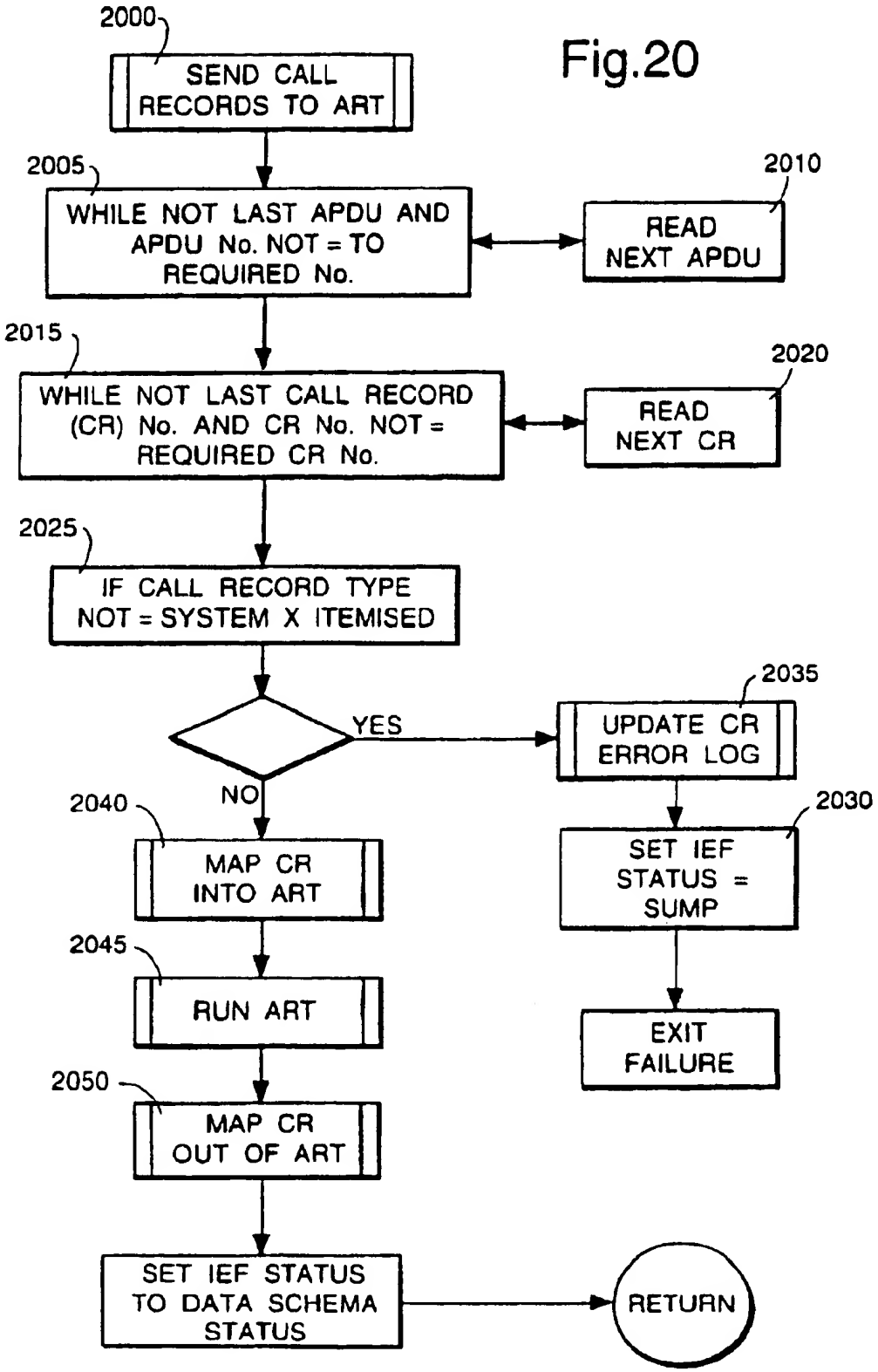

1901
START
1902
IS SUSPENSE SHUTDOWN ?
YES
STOP
1903
LOCATE EARLIEST PATTERN WHICH HAS BEEN FIXED BY NETWORK MANAGEMENT CENTRE
LOCATE NEXT EXCHANGE FILE CONTAINING THAT PATTERN
1904
1905
GET NEXT SUSPENDED CALL RECORD
1906
VALIDATE CALL RECORD
1907
NO
IS IT ANOTHER ROUTING FAILURE ?
1908
GO TO 1801
1909
GO TO 1501
1910
NO
ALL SUSPENDED CALL RECORDS FINISHED FOR EXCHANGE FILE ?
YES
1911
NO
ALL FILES COMPLETED FOR ROUTE PATTERN ?
YES

Fig.20


2000
SEND CALL RECORDS TO ART
2005
WHILE NOT LAST APDU AND APDU No. NOT = TO REQUIRED No.
2010
READ NEXT APDU
2015
WHILE NOT LAST CALL RECORD (CR) No. AND CR No. NOT = REQUIRED CR No.
2020
READ NEXT CR
2025
IF CALL RECORD TYPE NOT = SYSTEM X ITEMISED
YES
2035
UPDATE CR ERROR LOG
NO
2040
MAP CR INTO ART
2030
SET IEF STATUS = SUMP
2045
RUN ART
EXIT FAILURE
2050
MAP CR OUT OF ART
SET IEF STATUS TO DATA SCHEMA STATUS
RETURN

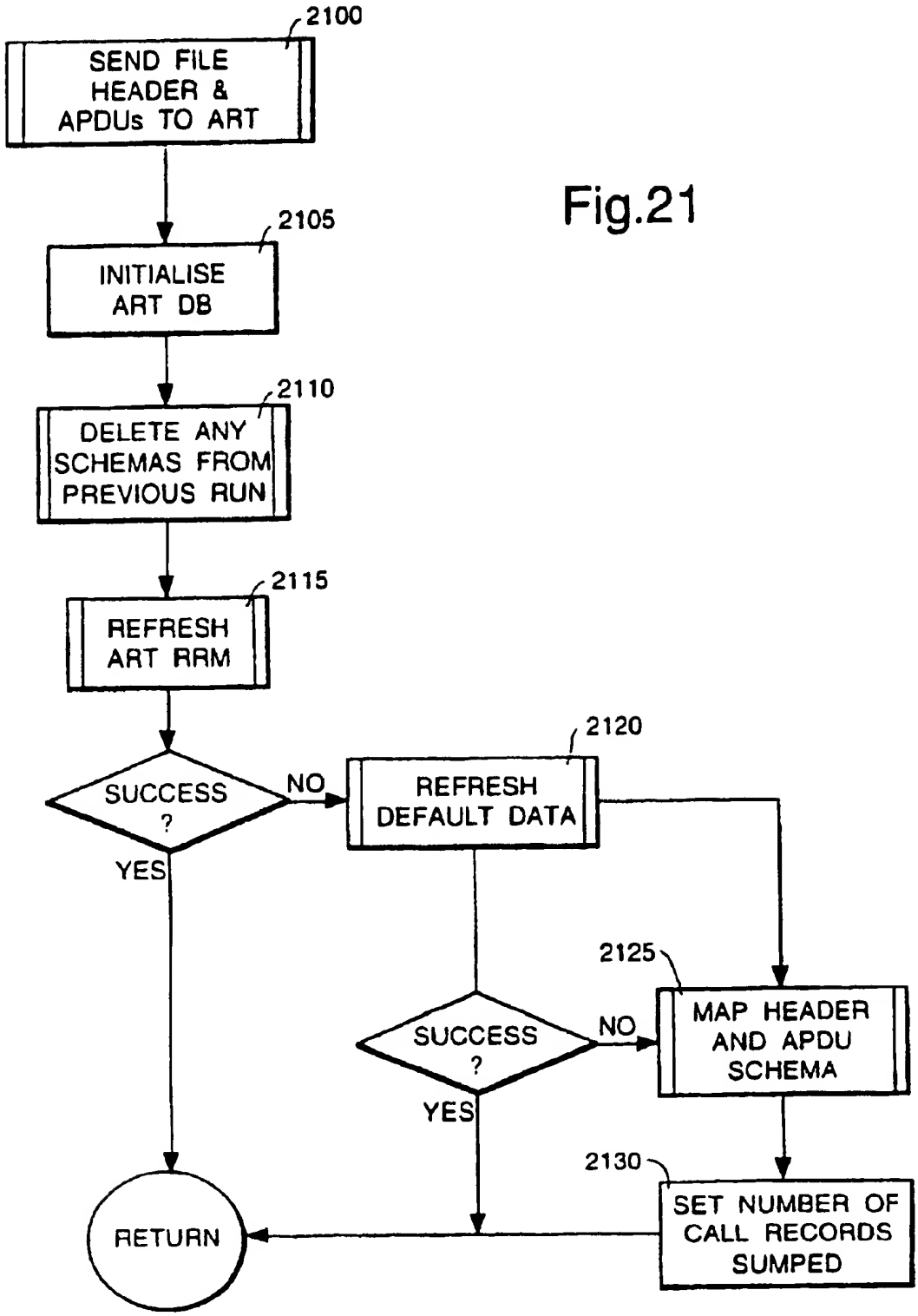
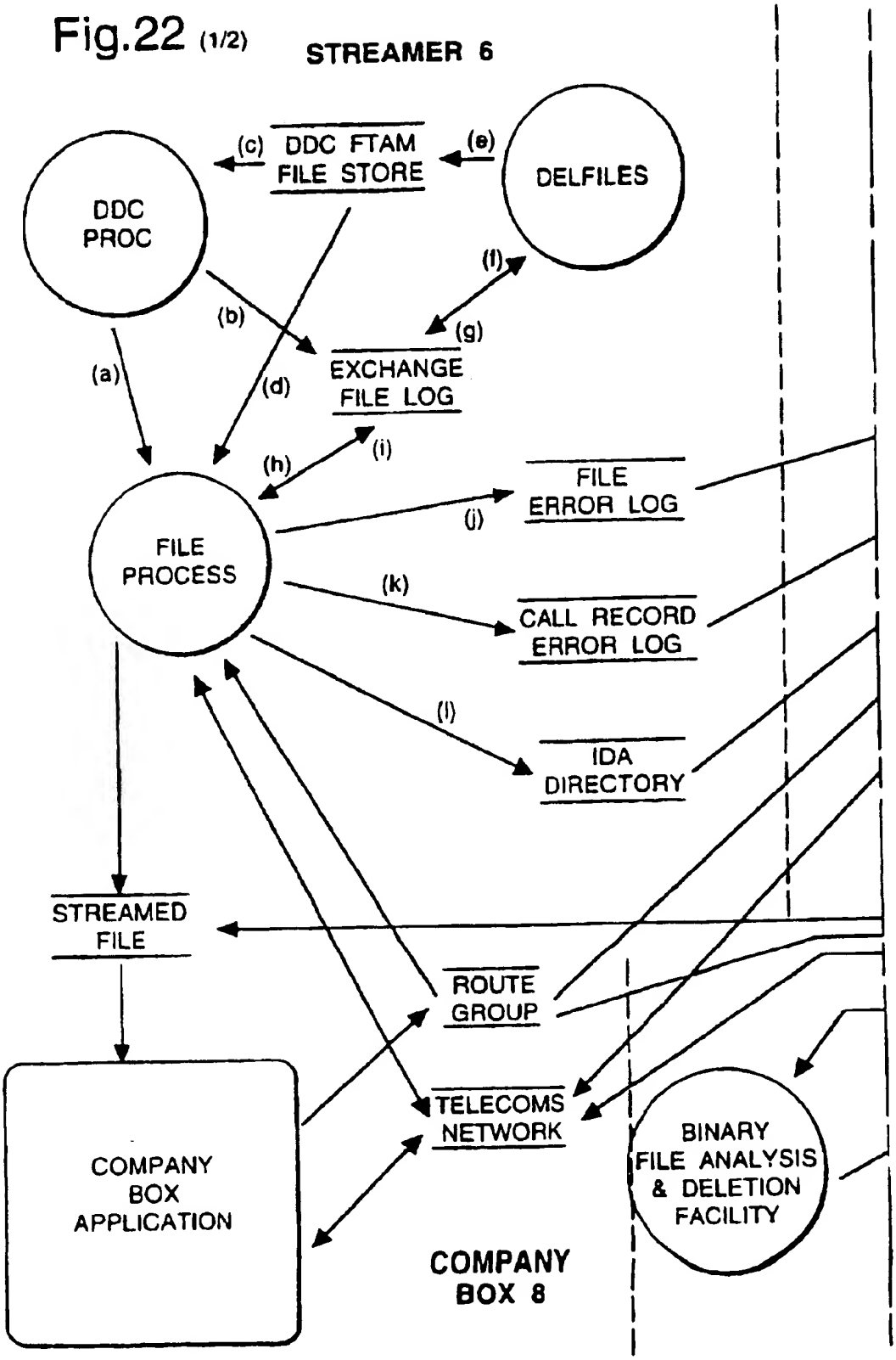

2100
SEND FILE HEADER & APDUs TO ART
2105
INITIALISE ART DB
2110
DELETE ANY SCHEMAS FROM PREVIOUS RUN
2115
REFRESH ART RRM
SUCCESS ?
NO
2120
REFRESH DEFAULT DATA
YES
2125
SUCCESS ?
NO
MAP HEADER AND APDU SCHEMA
YES
2130
SET NUMBER OF CALL RECORDS SUMPED
RETURN

Fig.21

Fig.22 (1/2)


STREAMER 6
DDC PROC
(c)
DDC FTAM FILE STORE
(e)
DELFILES
(f)
(b)
(g)
(a)
(d)
EXCHANGE FILE LOG
(h)
(i)
FILE ERROR LOG
(j)
FILE PROCESS
(k)
CALL RECORD ERROR LOG
(l)
IDA DIRECTORY
STREAMED FILE
ROUTE GROUP
TELECOMS NETWORK
BINARY FILE ANALYSIS & DELETION FACILITY
COMPANY BOX APPLICATION
COMPANY BOX 8

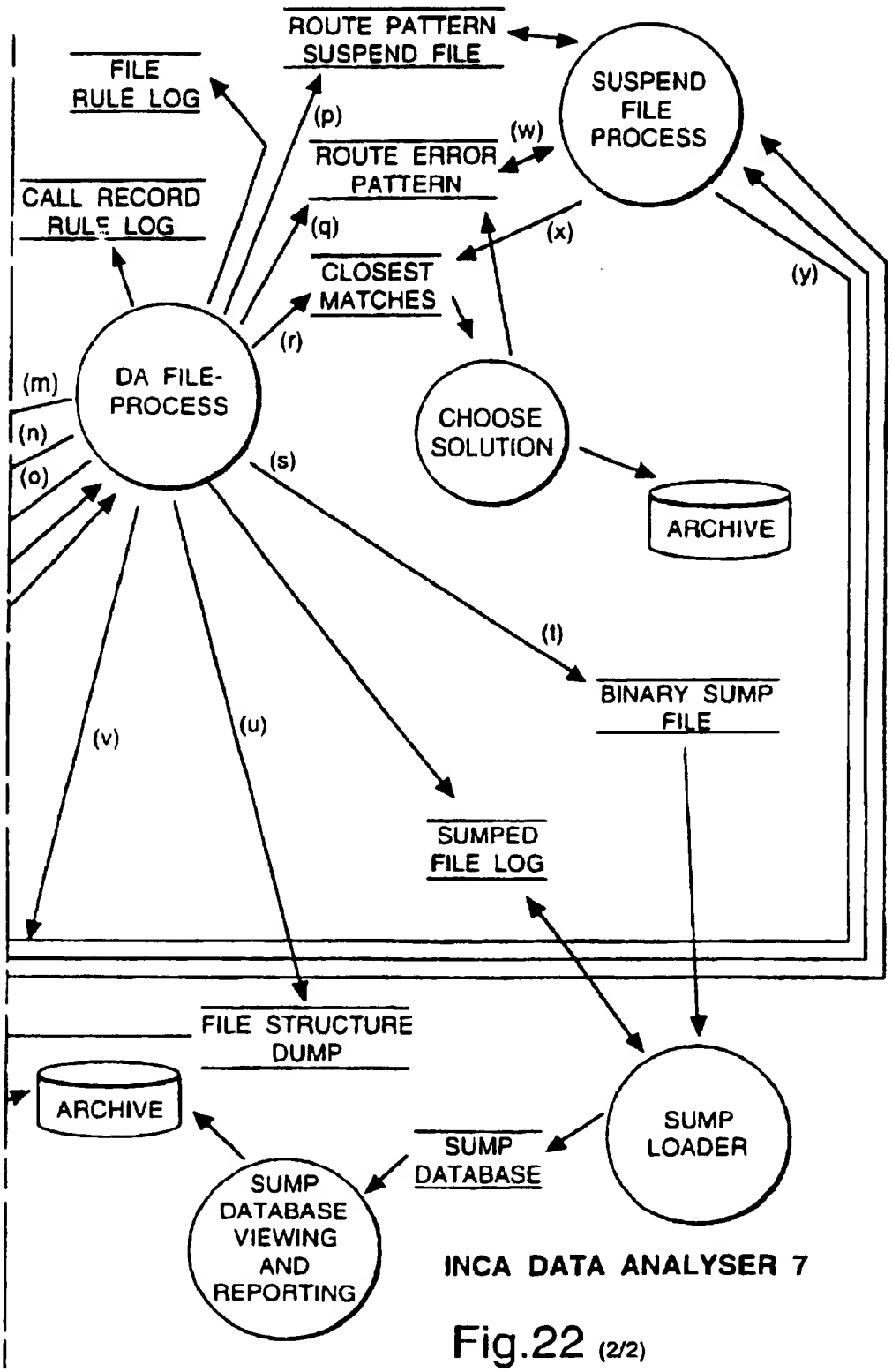

ROUTE PATTERN SUSPEND FILE
FILE RULE LOG
SUSPEND FILE PROCESS
(p)
(w)
ROUTE ERROR PATTERN
CALL RECORD RULE LOG
(q)
(x)
CLOSEST MATCHES
(y)
(r)
DA FILE-PROCESS
CHOOSE SOLUTION
(m)
(n)
(o)
(s)
ARCHIVE
(t)
BINARY SUMP FILE
(v)
(u)
SUMPED FILE LOG
FILE STRUCTURE DUMP
ARCHIVE
SUMP LOADER
SUMP DATABASE
SUMP DATABASE VIEWING AND REPORTING
INCA DATA ANALYSER 7

Fig.22 (2/2)

Fig.23

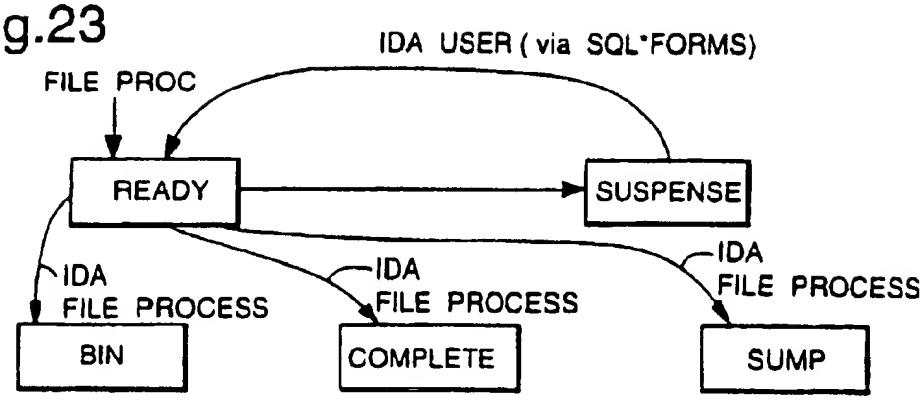
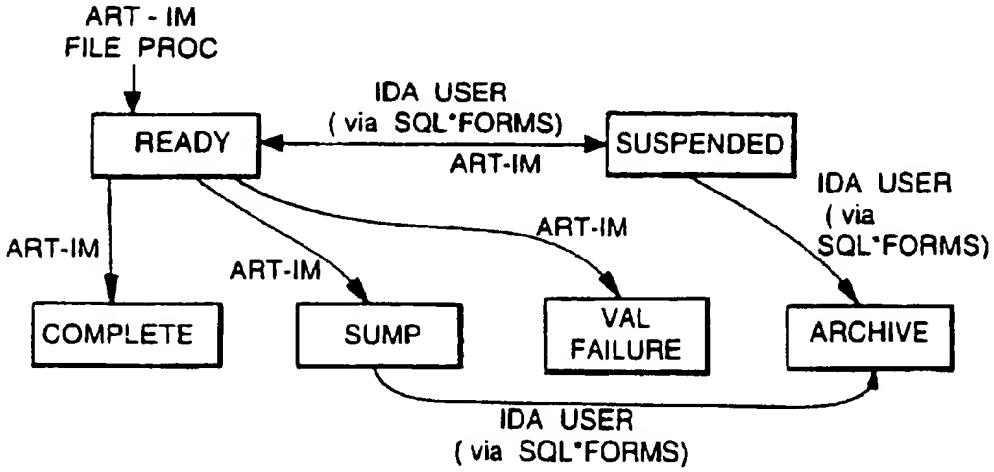


Fig.24



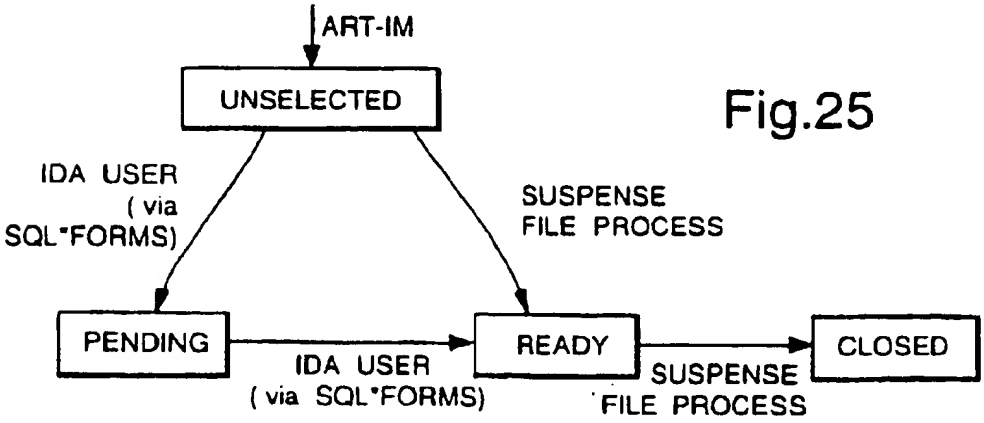
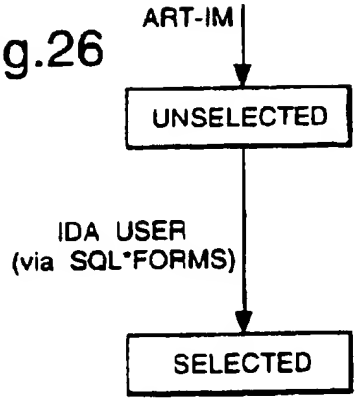


Fig.25

Fig.26



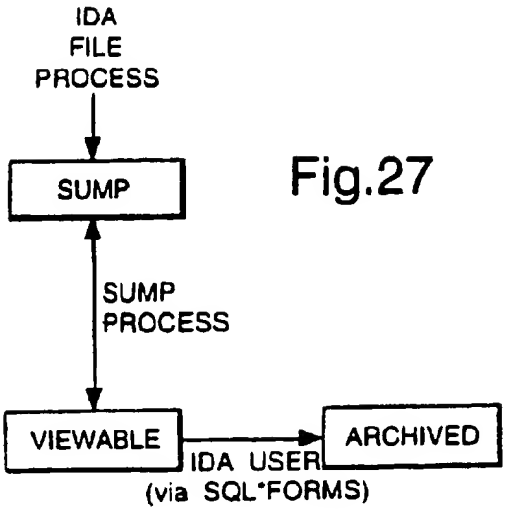


Fig.27

Fig.28

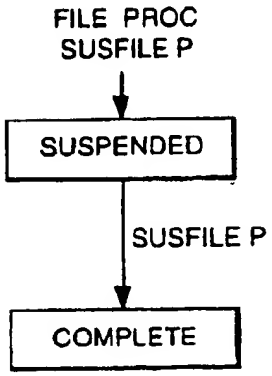


Fig.29

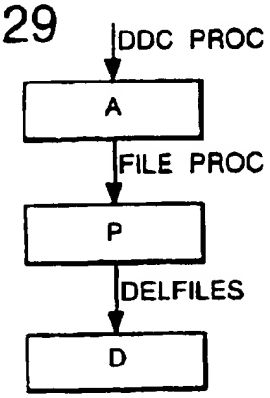


Fig.30

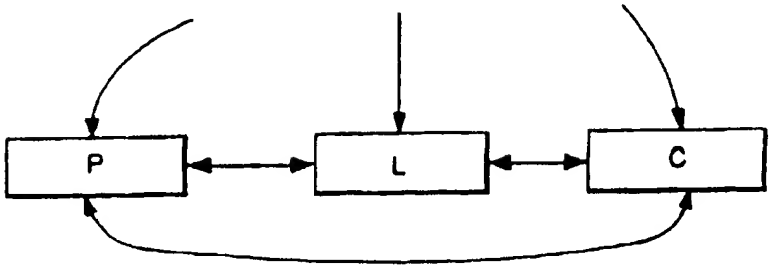


Fig.31

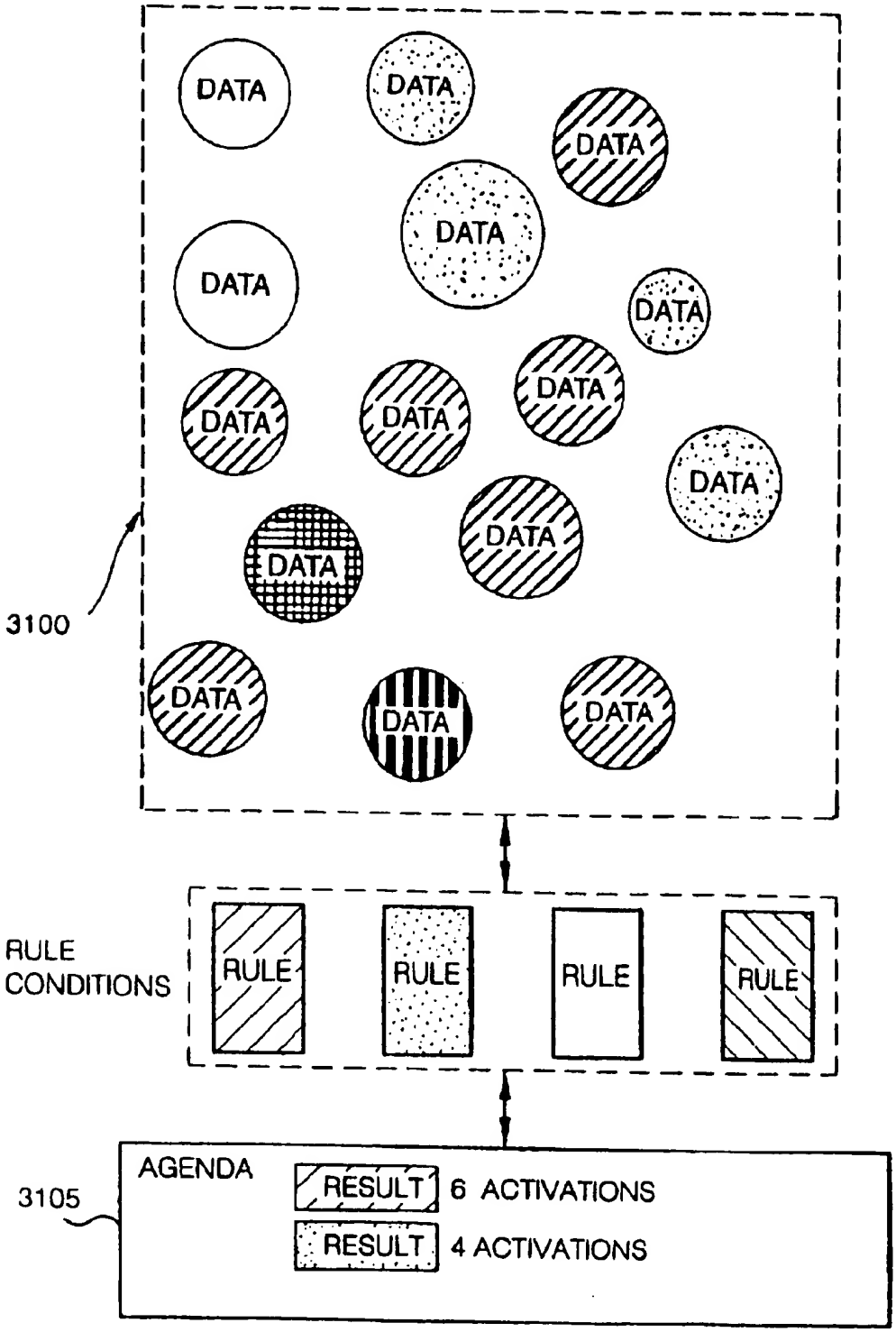
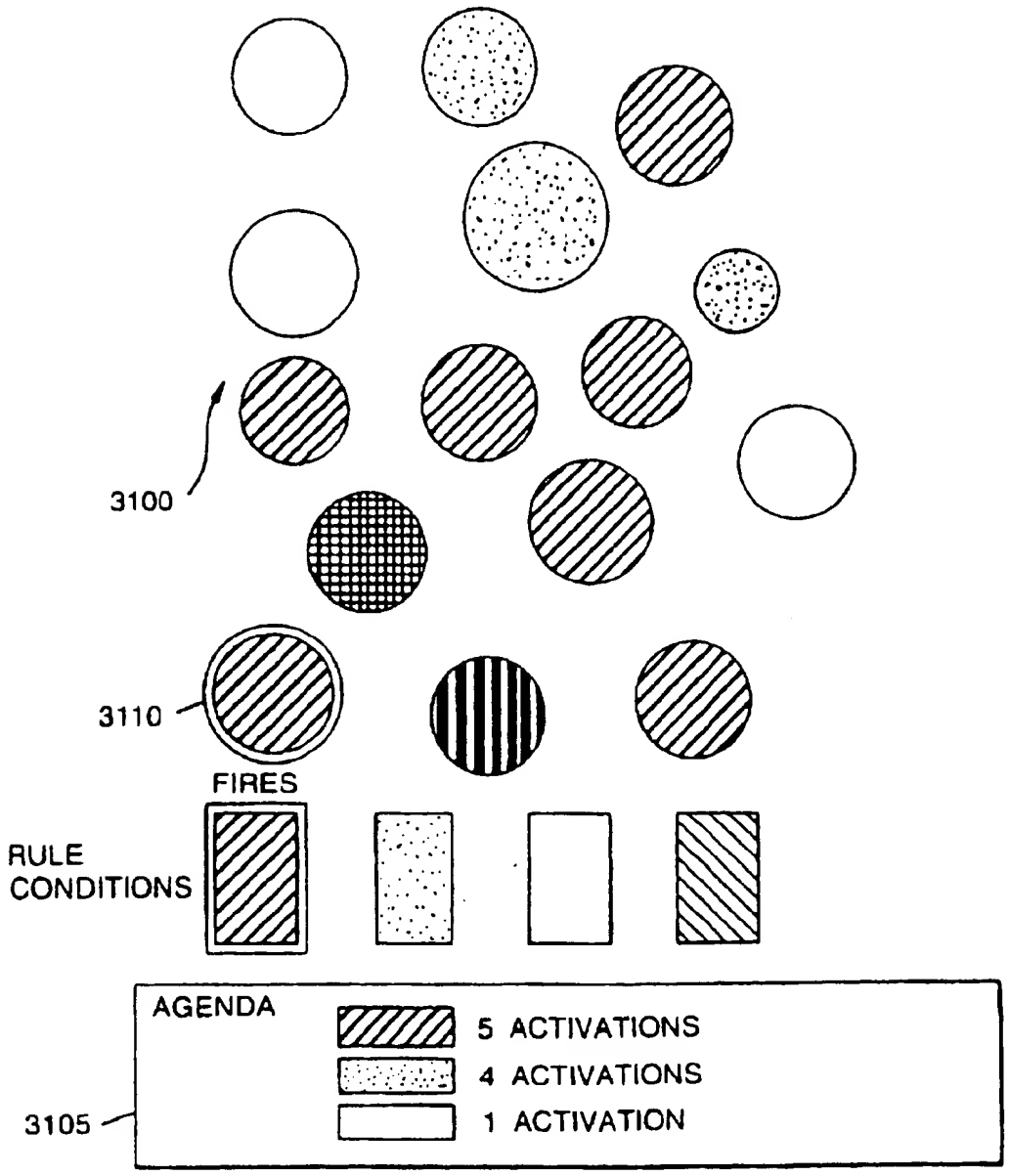

DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
3100
RULE CONDITIONS
RULE
RULE
RULE
RULE
AGENDA
3105
RESULT
6 ACTIVATIONS
RESULT
4 ACTIVATIONS

Fig.32


3100
3110
FIRES
RULE CONDITIONS
AGENDA
5 ACTIVATIONS
4 ACTIVATIONS
1 ACTIVATION
3105

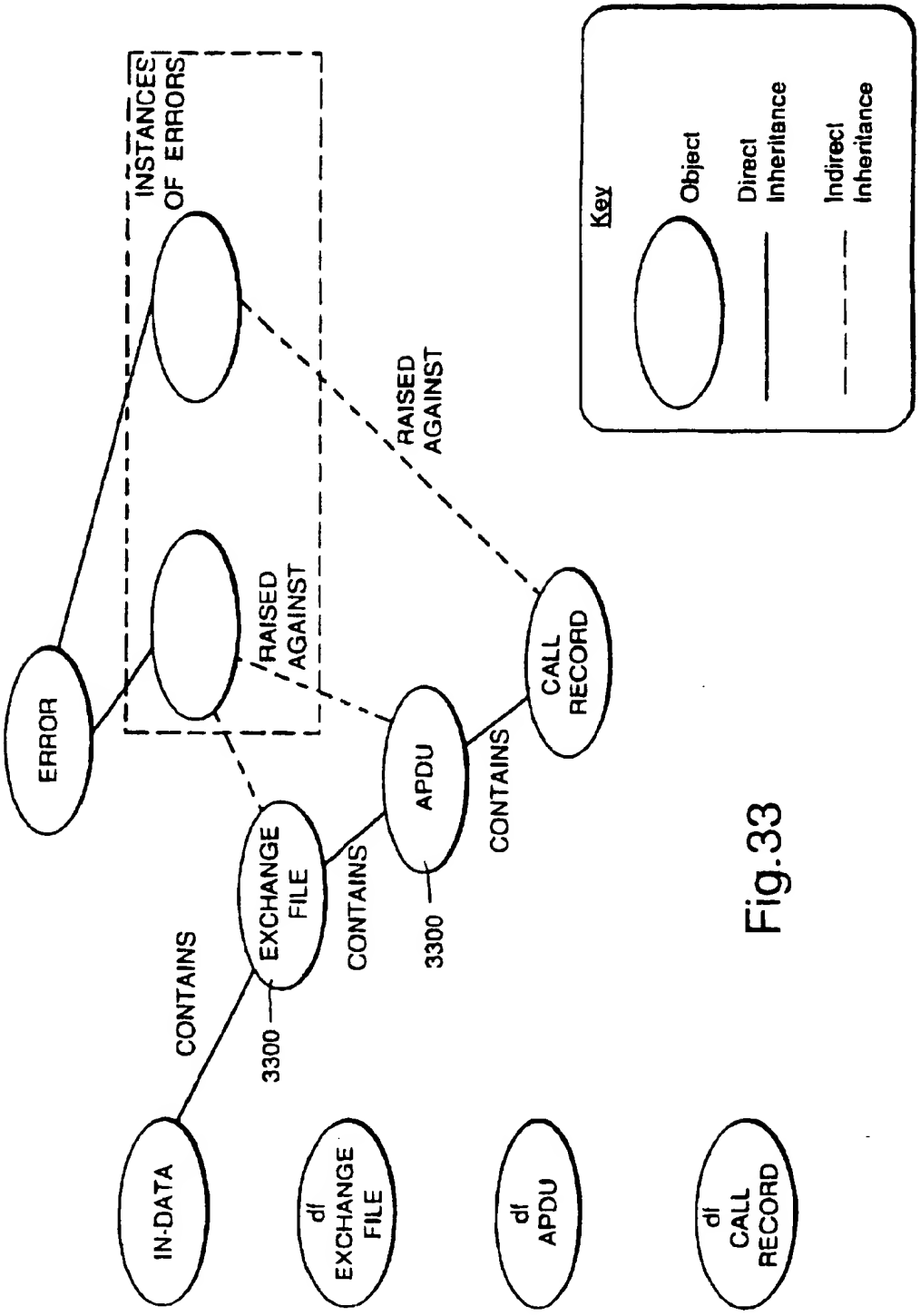

INSTANCES OF ERRORS
RAISED AGAINST
RAISED AGAINST
ERROR
CALL RECORD
APDU
CONTAINS
EXCHANGE FILE
CONTAINS
3300
3300
CONTAINS
IN-DATA
df EXCHANGE FILE
df APDU
df CALL RECORD
Key
Object
Direct Inheritance
Indirect Inheritance

Fig.33

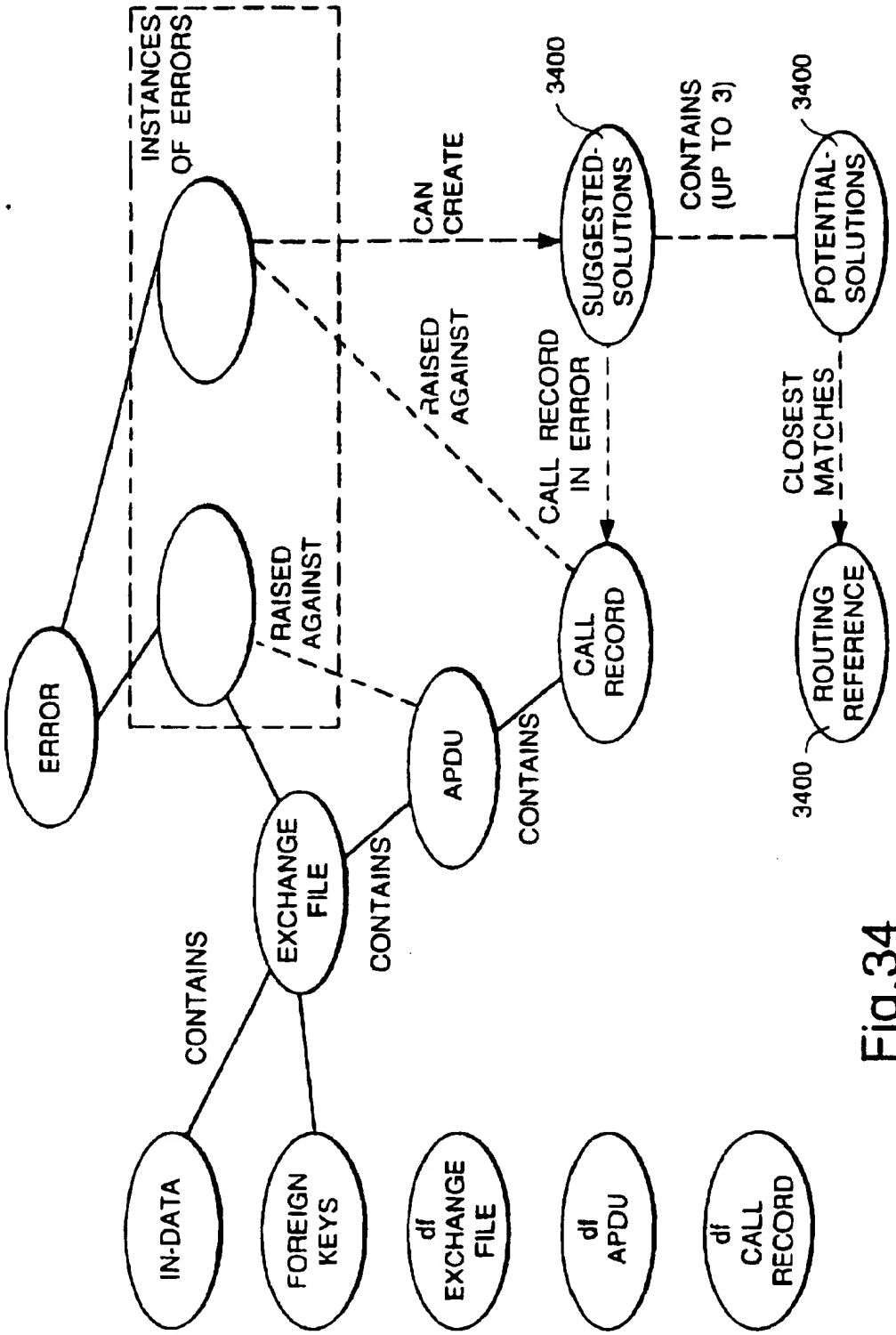

INSTANCES OF ERRORS
CAN CREATE
3400
SUGGESTED-SOLUTIONS
CONTAINS (UP TO 3)
3400
POTENTIAL-SOLUTIONS
RAISED AGAINST
CALL RECORD IN ERROR
CLOSEST MATCHES
RAISED AGAINST
ERROR
CALL RECORD
ROUTING REFERENCE
3400
APDU
CONTAINS
EXCHANGE FILE
CONTAINS
CONTAINS
IN-DATA
FOREIGN KEYS
df EXCHANGE FILE
df APDU
df CALL RECORD

Fig.34

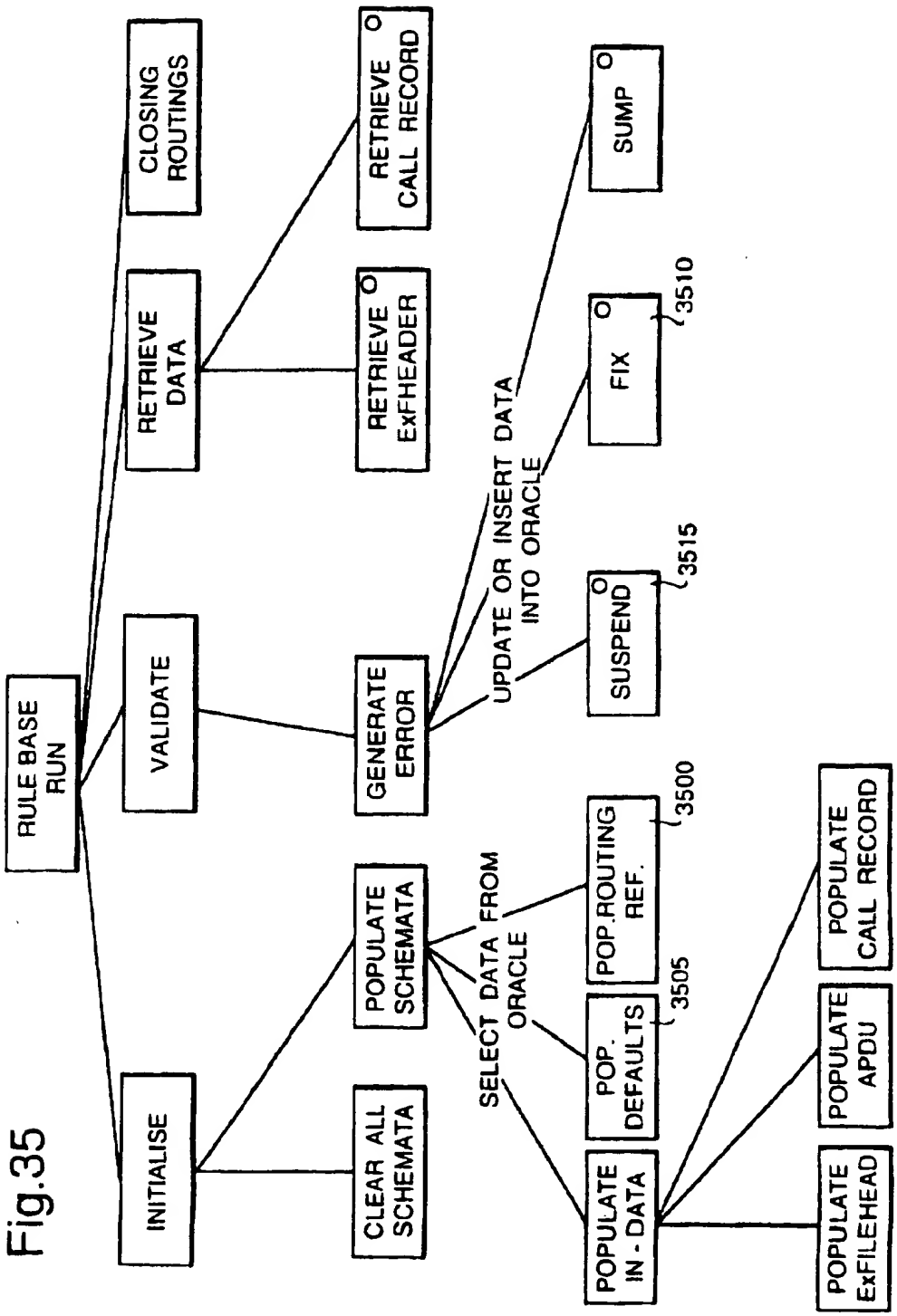

RULE BASE RUN
INITIALISE
CLEAR ALL SCHEMATA
POPULATE SCHEMATA
SELECT DATA FROM ORACLE
POP. DEFAULTS
3505
POP.ROUTING REF.
3500
POPULATE IN - DATA
POPULATE ExFILEHEAD
POPULATE APDU
POPULATE CALL RECORD
VALIDATE
GENERATE ERROR
UPDATE OR INSERT DATA INTO ORACLE
SUSPEND
3515
FIX
3510
SUMP
RETRIEVE DATA
RETRIEVE ExFHEADER
RETRIEVE CALL RECORD
CLOSING ROUTINGS

Fig.35

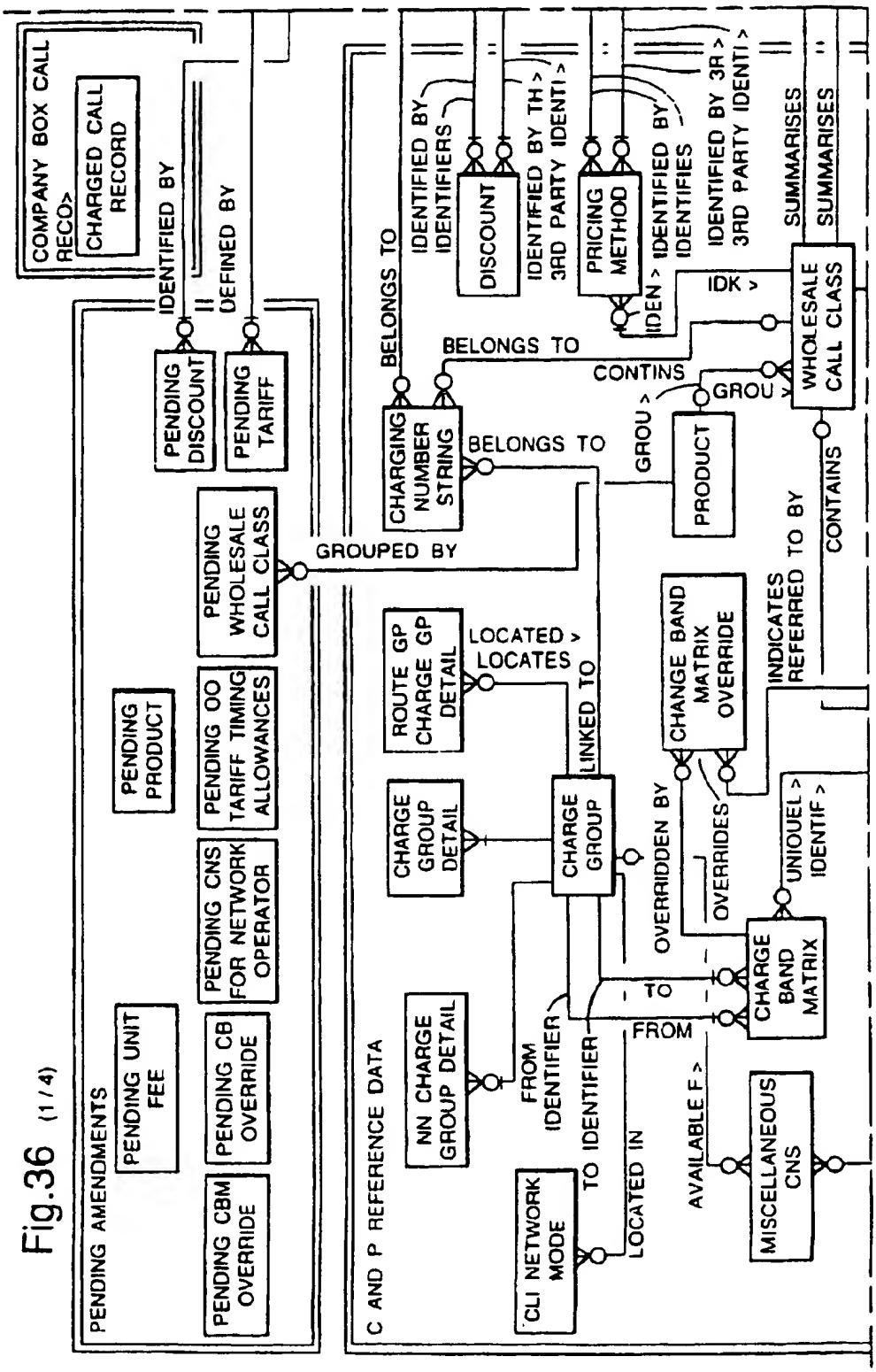

COMPANY BOX CALL RECO>
CHARGED CALL RECORD
IDENTIFIED BY
DEFINED BY
PENDING DISCOUNT
PENDING TARIFF
PENDING AMENDMENTS
PENDING UNIT FEE
PENDING PRODUCT
PENDING CBM OVERRIDE
PENDING CB OVERRIDE
PENDING CNS FOR NETWORK OPERATOR
PENDING OO TARIFF TIMING ALLOWANCES
PENDING WHOLESALE CALL CLASS
GROUPED BY
BELONGS TO
IDENTIFIED BY IDENTIFIERS
DISCOUNT
IDENTIFIED BY TH > 3RD PARTY IDENTI >
PRICING METHOD
IDEN > IDENTIFIED BY IDENTIFIES
IDENTIFIED BY 3R > 3RD PARTY IDENTI >
SUMMARISES
SUMMARISES
IDK >
WHOLESALE CALL CLASS
BELONGS TO
CONTINS
GROU >
GROU Y
CHARGING NUMBER STRING
BELONGS TO
PRODUCT
CONTAINS
INDICATES REFERRED TO BY
ROUTE GP CHARGE GP DETAIL
LOCATED > LOCATES
LINKED TO
CHANGE BAND MATRIX OVERRIDE
CHARGE GROUP DETAIL
CHARGE GROUP
OVERRIDDEN BY
OVERRIDES
UNIQUEL > IDENTIF >
C AND P REFERENCE DATA
NN CHARGE GROUP DETAIL
FROM IDENTIFIER
TO
FROM
CHARGE BAND MATRIX
CLI NETWORK MODE
TO IDENTIFIER
LOCATED IN
AVAILABLE F >
MISCELLANEOUS CNS

Fig.36 (1/4)

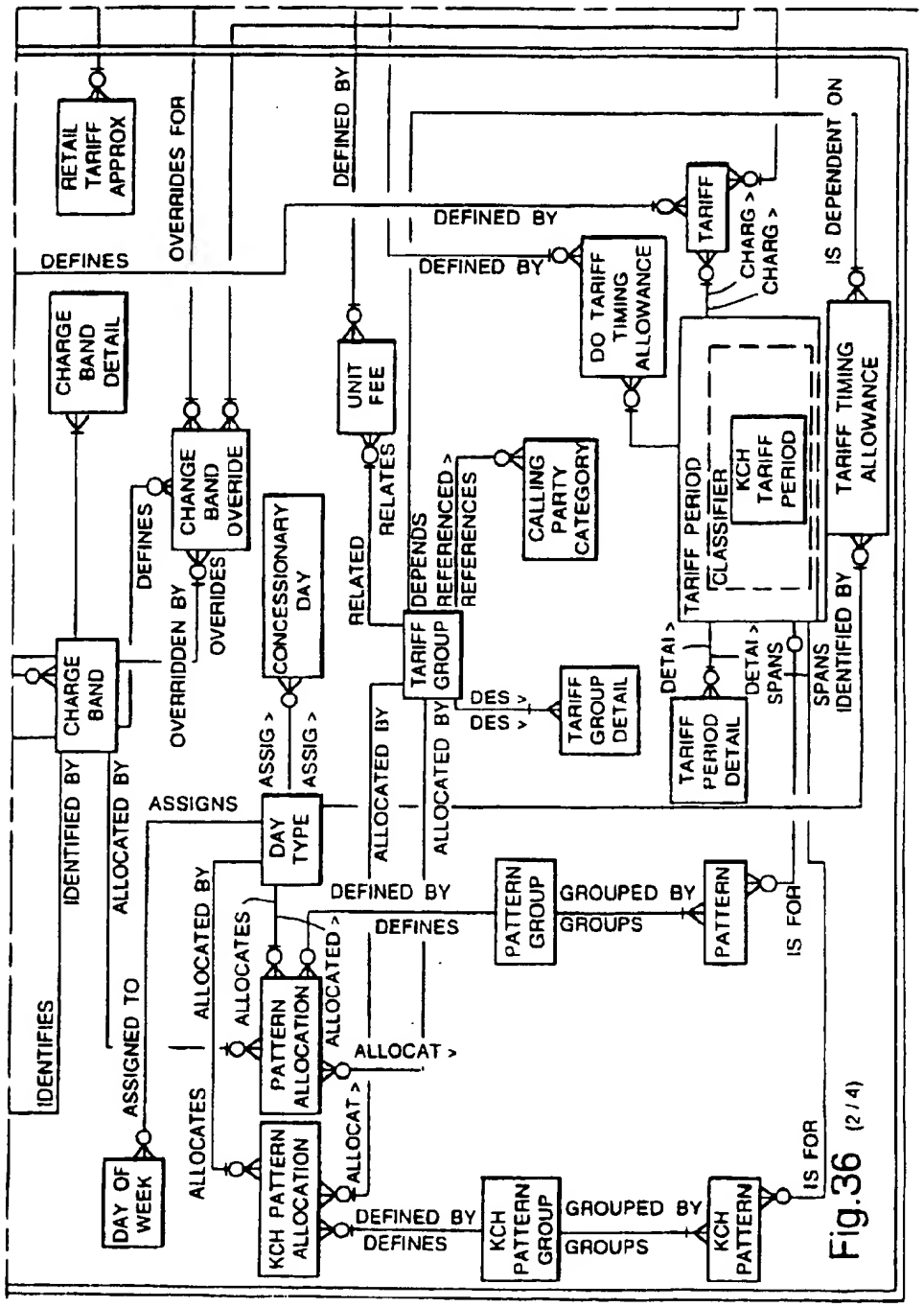

RETAIL TARIFF APPROX
OVERRIDES FOR
DEFINED BY
DEFINES
DEFINED BY
TARIFF
CHARG >
CHARG >
IS DEPENDENT ON
DEFINED BY
DO TARIFF TIMING ALLOWANCE
CHARGE BAND DETAIL
UNIT FEE
TARIFF PERIOD CLASSIFIER
KCH TARIFF PERIOD
TARIFF TIMING ALLOWANCE
DEFINES
CHANGE BAND OVERIDE
RELATES
RELATED
DEPENDS
REFERENCED >
REFERENCES
CALLING PARTY CATEGORY
OVERRIDDEN BY
OVERIDES
CONCESSIONARY DAY
TARIFF GROUP
DETAI >
SPANS
SPANS IDENTIFIED BY
CHARGE BAND
ASSIG >
ASSIG >
DES >
DES >
TARIFF GROUP DETAIL
TARIFF PERIOD DETAIL
IDENTIFIED BY
ALLOCATED BY
ALLOCATED BY
ALLOCATED BY
ASSIGNS
DAY TYPE
DEFINED BY
PATTERN GROUP
GROUPED BY
GROUPS
PATTERN
IS FOR
IDENTIFIES
ASSIGNED TO
ALLOCATES
ALLOCATED BY
ALLOCATES
PATTERN ALLOCATION
ALLOCATED >
DEFINES
ALLOCAT >
ALLOCAT >
DAY OF WEEK
KCH PATTERN ALLOCATION
DEFINED BY
DEFINES
KCH PATTERN GROUP
GROUPED BY
GROUPS
KCH PATTERN
IS FOR

Fig.36 (2/4)

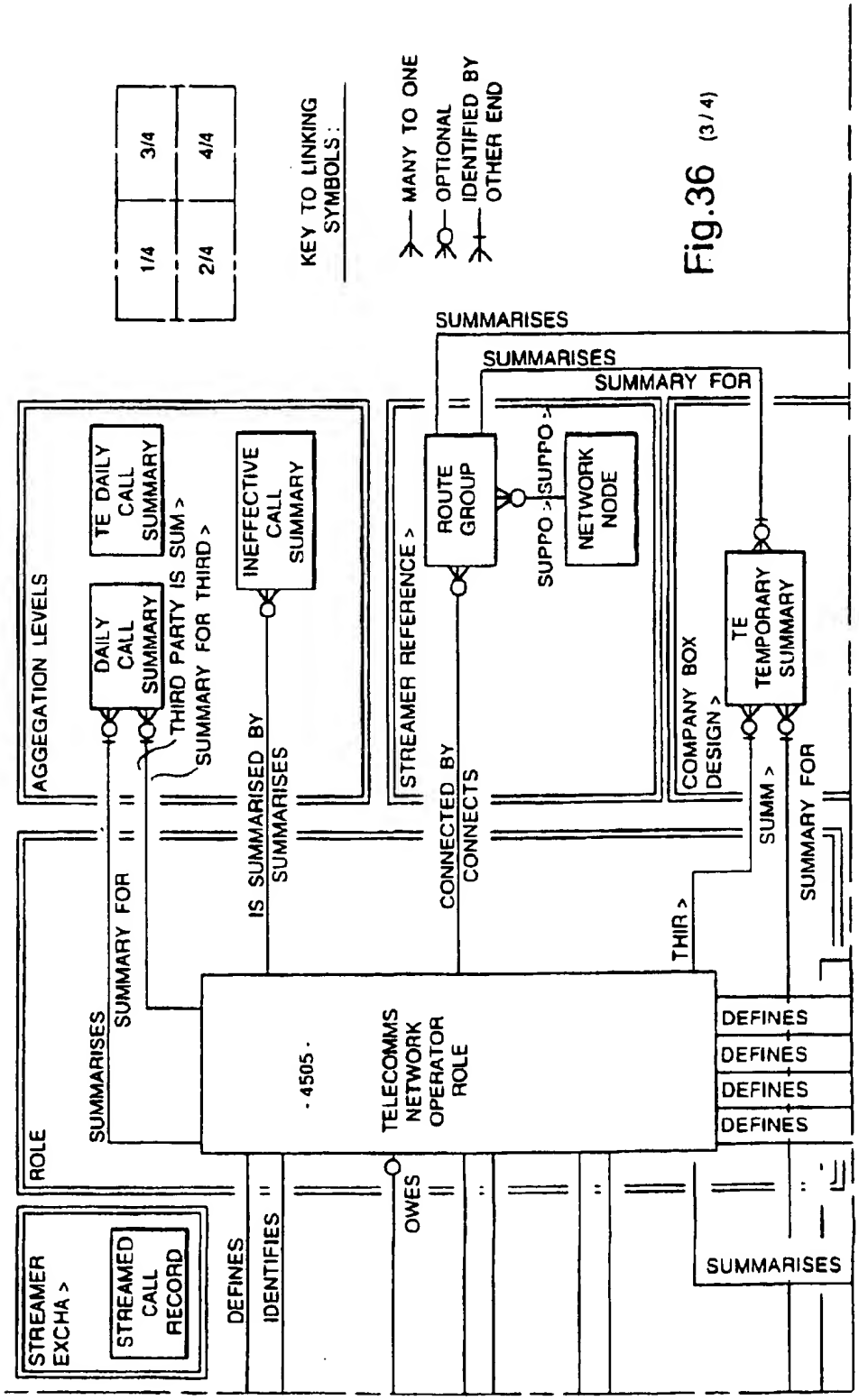

1/4
3/4
2/4
4/4
KEY TO LINKING SYMBOLS:
MANY TO ONE
OPTIONAL
IDENTIFIED BY OTHER END
SUMMARISES
SUMMARISES
SUMMARY FOR
AGGEGATION LEVELS
TE DAILY CALL SUMMARY
DAILY CALL SUMMARY
INEFFECTIVE CALL SUMMARY
THIRD PARTY IS SUM >
SUMMARY FOR THIRD >
IS SUMMARISED BY
SUMMARISES
STREAMER REFERENCE >
ROUTE GROUP
SUPPO >
SUPPO >
NETWORK NODE
COMPANY BOX
DESIGN >
TE TEMPORARY SUMMARY
CONNECTED BY
CONNECTS
SUMM >
SUMMARY FOR
THIR >
ROLE
SUMMARISES
SUMMARY FOR
- 4505 -
TELECOMMS NETWORK OPERATOR ROLE
DEFINES
DEFINES
DEFINES
DEFINES
STREAMER EXCHA >
STREAMED CALL RECORD
DEFINES
IDENTIFIES
OWES
SUMMARISES
Fig.36 (3/4)

Fig.36 (4 / 4)

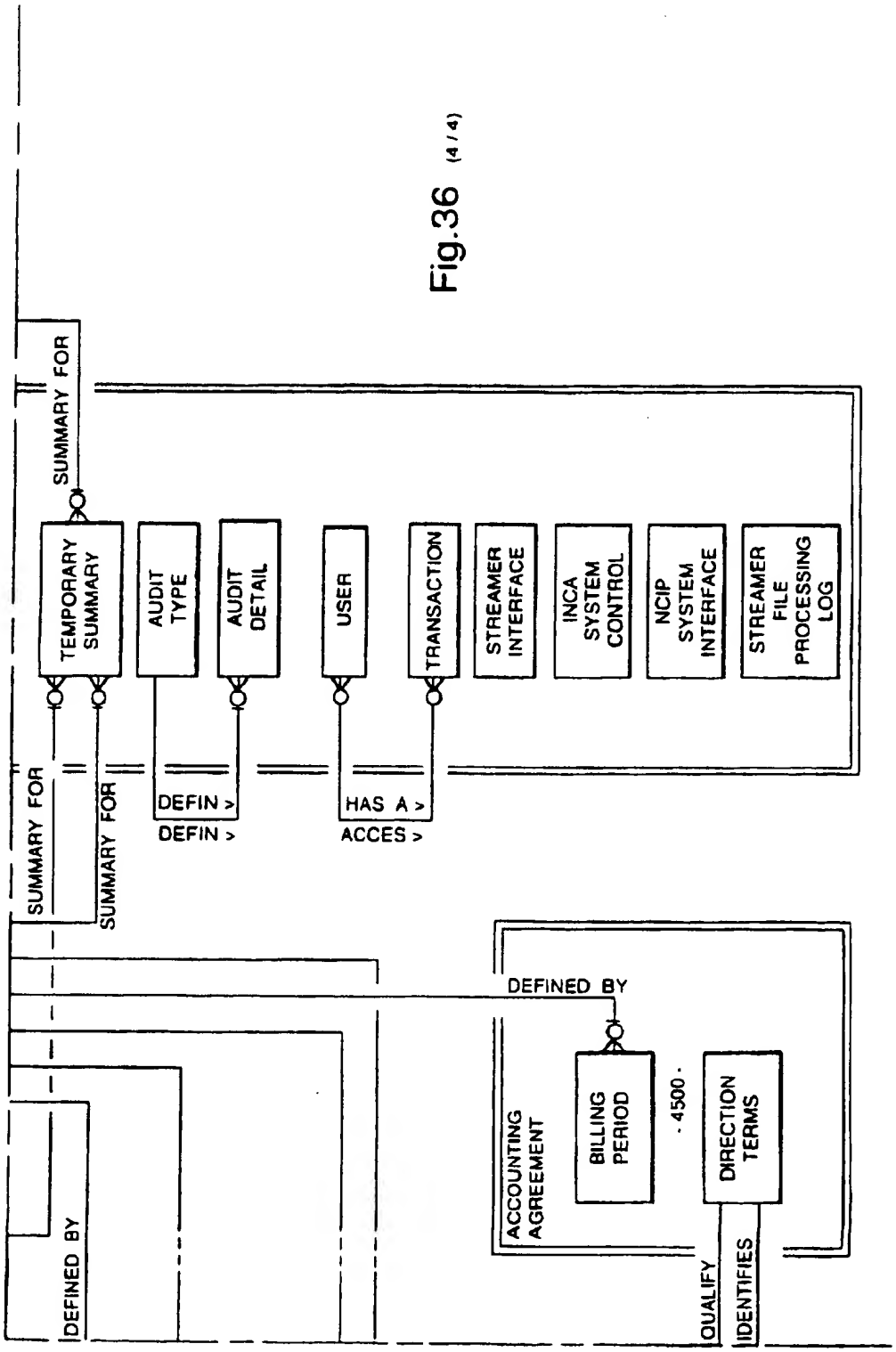

SUMMARY FOR
TEMPORARY SUMMARY
AUDIT TYPE
AUDIT DETAIL
USER
TRANSACTION
STREAMER INTERFACE
INCA SYSTEM CONTROL
NCIP SYSTEM INTERFACE
STREAMER FILE PROCESSING LOG
SUMMARY FOR
SUMMARY FOR
DEFIN >
DEFIN >
HAS A >
ACCES >
DEFINED BY
ACCOUNTING AGREEMENT
BILLING PERIOD
· 4500 ·
DIRECTION TERMS
DEFINED BY
QUALIFY
IDENTIFIES

Fig.37

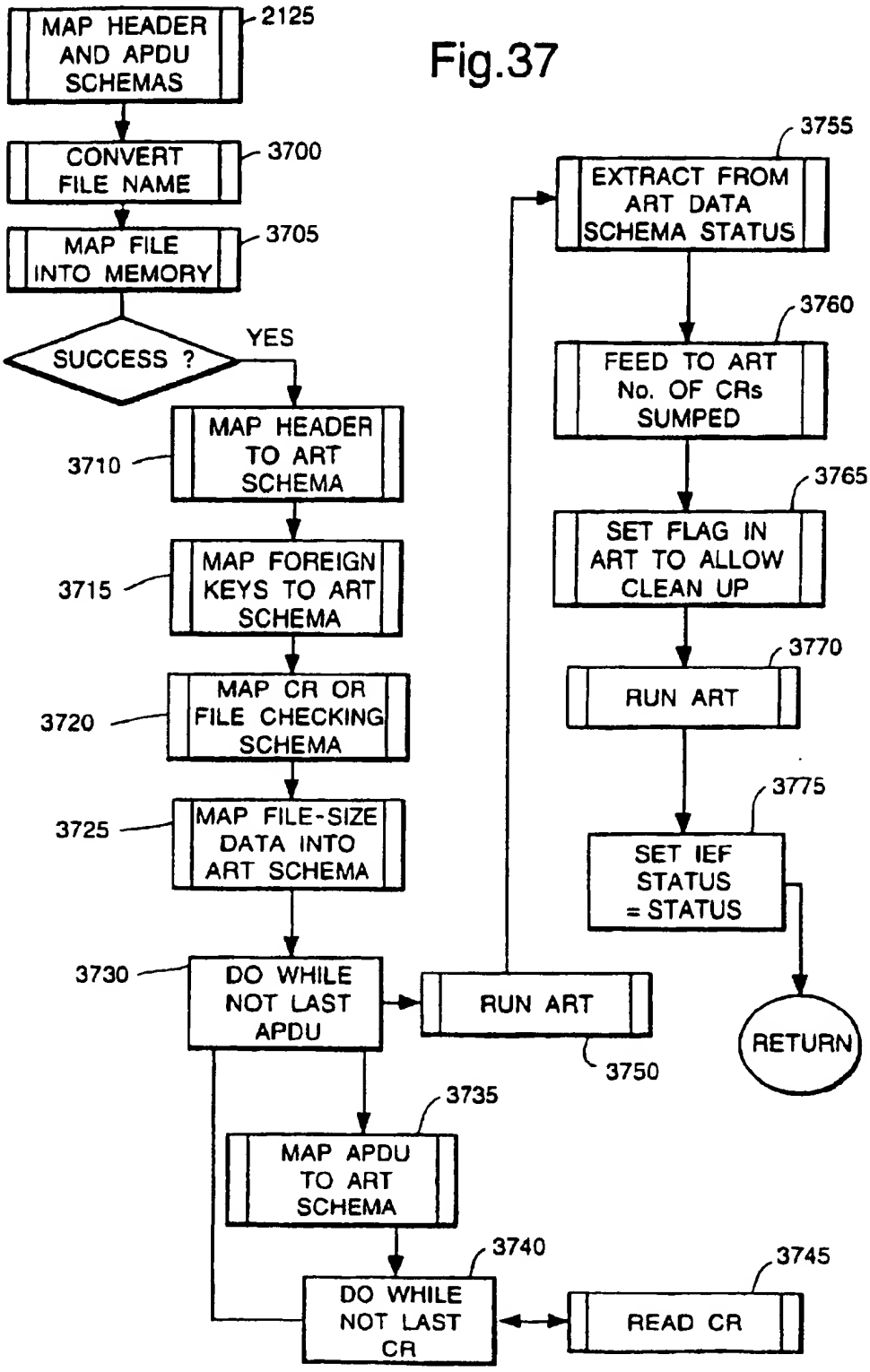

MAP HEADER AND APDU SCHEMAS
2125
CONVERT FILE NAME
3700
MAP FILE INTO MEMORY
3705
SUCCESS ?
YES
MAP HEADER TO ART SCHEMA
3710
MAP FOREIGN KEYS TO ART SCHEMA
3715
MAP CR OR FILE CHECKING SCHEMA
3720
MAP FILE-SIZE DATA INTO ART SCHEMA
3725
DO WHILE NOT LAST APDU
3730
RUN ART
3750
EXTRACT FROM ART DATA SCHEMA STATUS
3755
FEED TO ART No. OF CRs SUMPED
3760
SET FLAG IN ART TO ALLOW CLEAN UP
3765
RUN ART
3770
SET IEF STATUS = STATUS
3775
RETURN
MAP APDU TO ART SCHEMA
3735
DO WHILE NOT LAST CR
3740
READ CR
3745

Fig.38

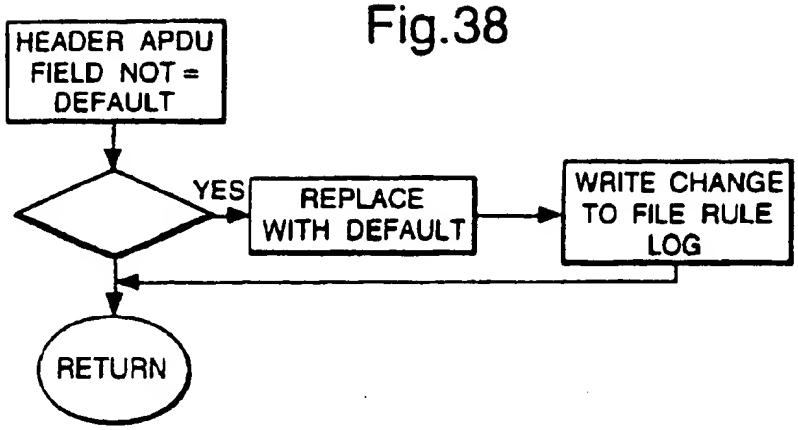


Fig.39

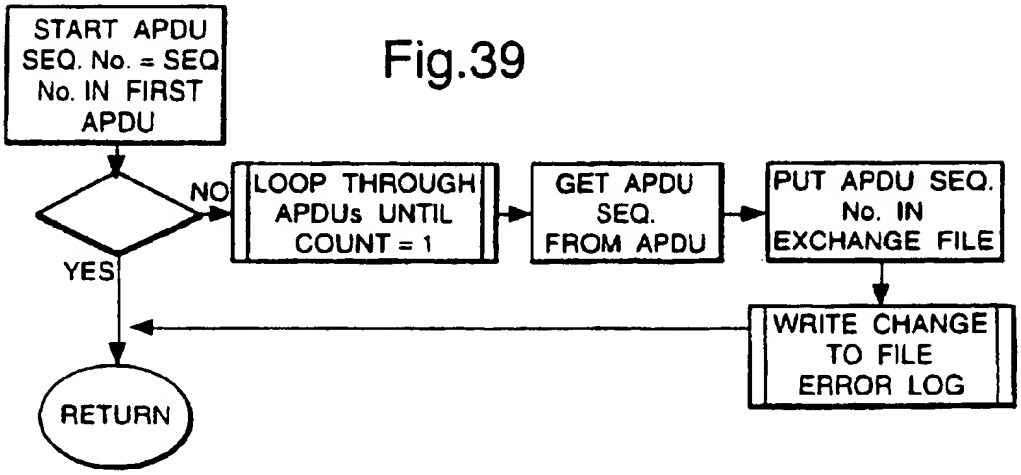


Fig.40

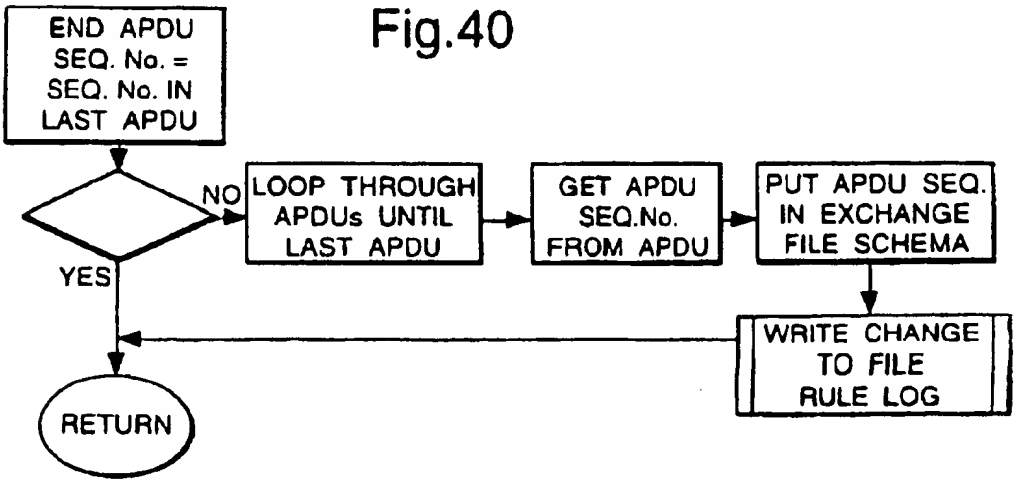


Fig.41

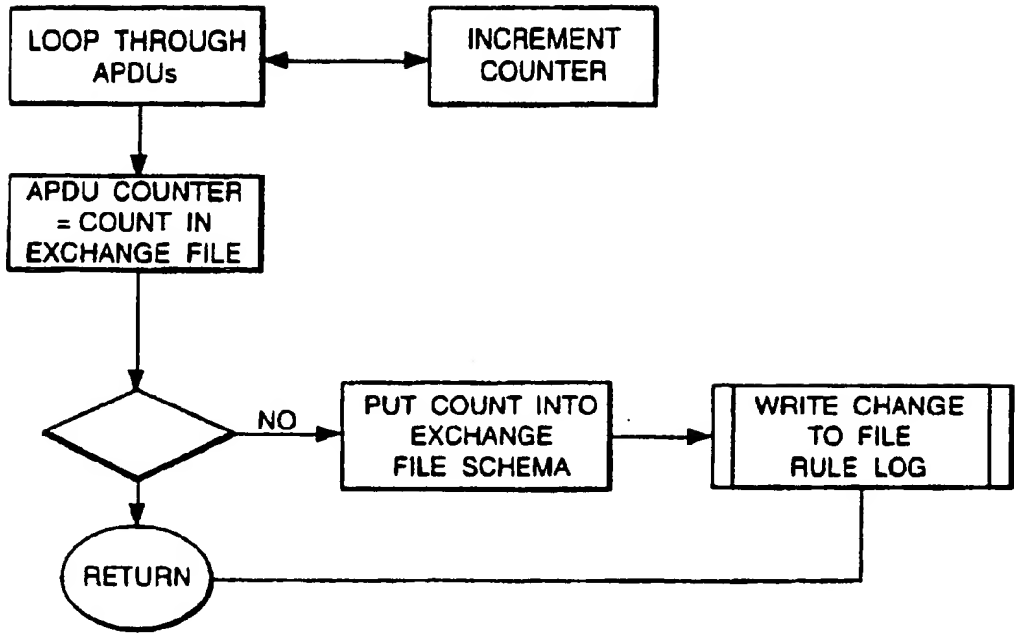

LOOP THROUGH APDUs
INCREMENT COUNTER
APDU COUNTER = COUNT IN EXCHANGE FILE
NO
PUT COUNT INTO EXCHANGE FILE SCHEMA
WRITE CHANGE TO FILE RULE LOG
RETURN

Fig.42

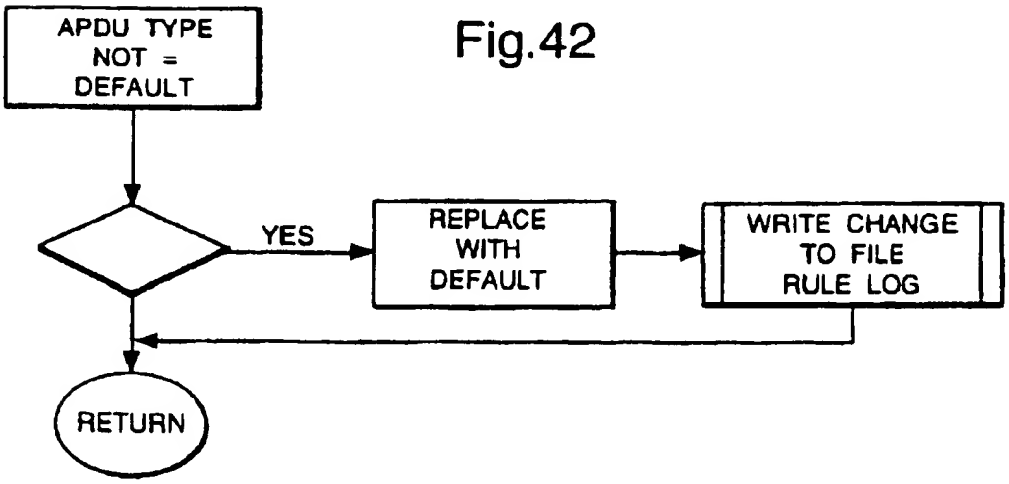
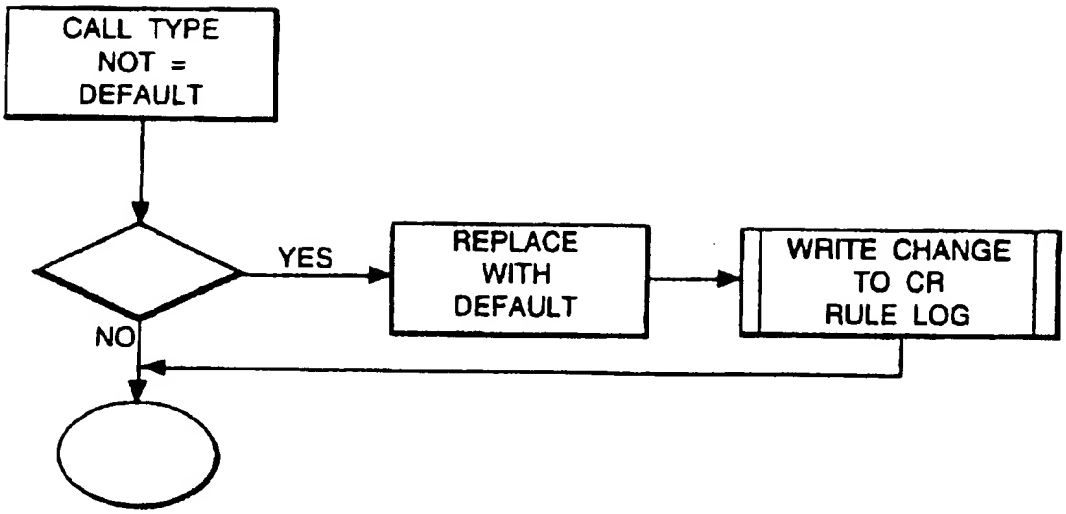

APDU TYPE NOT = DEFAULT
YES
REPLACE WITH DEFAULT
WRITE CHANGE TO FILE RULE LOG
RETURN

Fig.43


CALL TYPE NOT = DEFAULT
YES
REPLACE WITH DEFAULT
WRITE CHANGE TO CR RULE LOG
NO

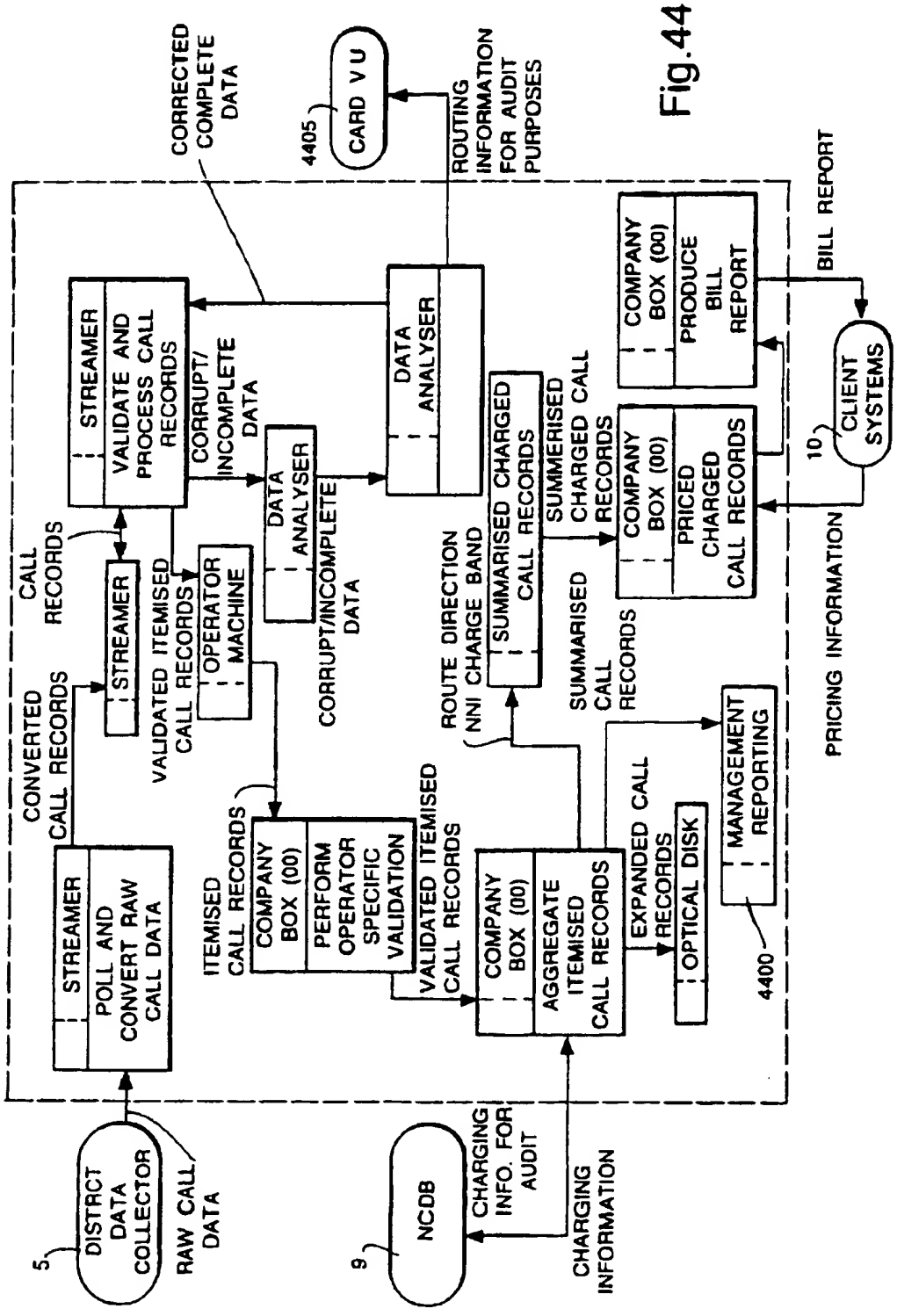

DISTRCT DATA COLLECTOR
5
RAW CALL DATA
NCDB
9
CHARGING INFO. FOR AUDIT
CHARGING INFORMATION
STREAMER
POLL AND CONVERT RAW CALL DATA
CONVERTED CALL RECORDS
STREAMER
CALL RECORDS
STREAMER
VALIDATE AND PROCESS CALL RECORDS
CORRECTED COMPLETE DATA
CORRUPT/ INCOMPLETE DATA
VALIDATED ITEMISED CALL RECORDS
OPERATOR MACHINE
ITEMISED CALL RECORDS
COMPANY BOX (00)
PERFORM OPERATOR SPECIFIC VALIDATION
VALIDATED ITEMISED CALL RECORDS
DATA ANALYSER
CORRUPT/INCOMPLETE DATA
DATA ANALYSER
4405
CARD V U
ROUTING INFORMATION FOR AUDIT PURPOSES
ROUTE DIRECTION NNI CHARGE BAND
SUMMARISED CHARGED CALL RECORDS
SUMMERISED CHARGED CALL RECORDS
COMPANY BOX (00)
AGGREGATE ITEMISED CALL RECORDS
EXPANDED CALL RECORDS
OPTICAL DISK
4400
SUMMARISED CALL RECORDS
MANAGEMENT REPORTING
COMPANY BOX (00)
PRODUCE BILL REPORT
BILL REPORT
COMPANY BOX (00)
PRICED CHARGED CALL RECORDS
PRICING INFORMATION
10
CLIENT SYSTEMS

Fig.44

DATA PROCESSING SYSTEM FOR COMMUNICATIONS NETWORK

Matter enclosed in heavy brackets [ ] appears in the original patent but forms no part of this reissue specification; matter printed in italics indicates the additions made by reissue.

RELATED APPLICATION

This application is related to my copending commonly assigned application Ser. No. 08/392,975 filed Mar. 6, 1995.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a data system for data collection and processing in multi network communications.

2. Related Art

Where communication instances, for instance telephone calls or data transfers, occur within a single network, it is known to log and process data related to those communication instances. Commonly, in a public switched telephone network (PSTN), data will be collected concerning call duration, and processed with respect to at least time of day and call type, so that the network operator can generate an item on a bill destined for the subscriber who initiated a call.

Over recent years, the data systems for PSTNs have necessarily become increasingly complex as the choice of service and call type available to subscribers has greatly increased. For instances, with the introduction of 0800 numbers, it is no longer the initiating subscriber who will be billed. Many more complicated services are already being trialled, or available, on PSTNs, such as call forwarding where a call initiated by a first subscriber to a selected number is forwarded automatically by the network to a different number, the different in cost being borne by the receiving subscriber.

Another aspect of communication networks which is in the course of considerable change is the multiplicity of network operators in existence. In the past, PSTNs have been run primarily by government organizations as part of the national infra structure. Nowadays and increasingly, privatization of the PSTNs and the relaxation of regulatory monopolies means that there are many more network operators available to the subscriber and these network operators must, for practical reasons, provide inter network connection. This means that a network operator must take into account not only communication instances arising in their own network or in a limited number of inter-connected networks of independent but similar administrations, but also communication instances arising in a theoretically very large number of competing networks of different types and providing a wide variety of services to subscribers.

It is, therefore, of increasing importance that data be collected and processed in connection with communication instances arising outside an operator's network but terminating in or simply crossing the operator'network.

When calls pass through the network of more than one operator, price and charging agreements between operators for the carriage of each other's calls come into play. Such arrangements can vary from the simple Sender Keeps All (SKA) arrangement to complex pricing formulae.

It has been an established practice between separate network operators or administrations, in telecommunications, that call data would be collected by the administration responsible for the network in which a call

arises. If that call then terminates in a second network, the administration concerned with the second network relies on the data collected by the administration responsible for the first network, for instance for accounting purposes, However, the telecommunications environment is changing quickly, politically as well as technically. With the advent of greater competition, it is increasingly attractive to network administrations to monitor not only traffic arising in their own network but also traffic arising elsewhere but crossing or terminating in their own network. If the network in which traffic arises belongs to a competing operator or administration, it is desirable that it is at least possible to cross check the competing operator's accounts.

In known arrangements, data collection points concerning calls in a PSTN have been at local exchanges of a network since the local exchange picks up traffic as it arises. This arrangement, however, does not provide for data collection with respect to inter-network traffic. Even were there to be data collection points to collect data on calls incoming to a network, the logistics involved in processing such data to any level of detail are daunting. For instance, it has been estimated that calls incoming to the PSTN operated in Britain by British Telecommunications pic (BT) from other network administrations including the Isle of Man and the Cellnet cellular network totalled 15.4 million calls per day in the twelve months prior to March 1992. This figure is expected to increase to the order of 27 million calls a day in the year prior to March 1995. Taking all call instances into account, including those arising within the BT PSTN, 60 million call instances per day have been predicted for 1995.

SUMMARY OF THE INVENTION

In spike of the very large quantity of data involved, it has been found possible in making the present invention to design a process for collecting and processing data relating to calls incoming to a major telecommunications network, the British PSTN, which can produce an output in sufficient detail to allow the associated network administration to generate account information which not only can be allocated to outside network administrations appropriately, but also supports itemized billing. That is, the account information can be broken down in sufficient detail even to identify individual calls, so far as they fulfill preselected criteria, in the manner of itemized billing currently available in the national billing system for the British PSTN from British Telecommunications plc.

According to a first aspect of the present invention, there is provided a process for collecting and processing data concerning communication instances in a first communications network, wherein the network includes at least one point of connection, either directly or indirectly, to as second communications network, by means of which point of connection a communication instance arising in said second network can be transmitted into, and either cross or terminate in, said first network, the process comprising the steps of:

i) collecting data at a data access point at said point of connection, said data concerning a communication instance arising in said second network and comprising route information and at least one parameter measurement, or example duration, with respect to said communication instance;

ii) transmitting said data into a data processing system; and

iii) processing said data.

By collecting the data at a point of connection between the first network and another network, it becomes available to

the administration associated with the first network to obtain first hand information about communication instances incoming to the first network, and thus potentially to cross check data provided by other network operators or administrators.

According to a second aspect of the present invention, there is provided a data processing arrangement, for processing data collected in a PSTN at a point of connection with another network, the arrangement comprising:

i) a data input for inputting data concerning communication instances from a communication network, said data comprising at least one of a plurality of sort characteristics;

ii) verifying means for checking the integrity and sufficiency of data received at the data input;

iii) a data analyzer for analyzing data rejected by the verifying means, and for submitting amended or default data to the verifying means;

iv) pricing means for pricing verified data which has been output by the verifying means, in accordance with updatable reference information; and

v) output means for outputting priced, verified data from the pricing means into memory locations, each memory location being dedicated to data relevant to one or more of said sort characteristics.

Preferably, the pricing means can also validate data, and output errored data to a data analyzer, which may be the above data analyzer or a different one, so that data which has been corrupted can potentially be reformatted, or otherwise corrected, and, therefore, re-entered to the system as a valid record of a communication instance.

It may also (or alternatively) be that this further data analysis step is used to analyze the data with respect to a different type of fault. For instance, data analysis carried out on errored data which has been located by the verifying means might be errored principally in respect of format and routing information while the errored data from the pricing means might be errored principally in respect of pricing information.

The sort characteristics will typically be such that the memory locations can hold data relevant to communication instances which will be billable to a common accounting entity, for instance, arising in a common respective communications network.

The sort characteristics might be applied at any one of several stages of the data processing arrangement described above. However, in a PSTN for example, the nature of errored data usually arising makes it preferable to provide sorting means between (iii), the data analyzer associated with the verifying means, and (iv), the pricing means. The pricing means therefore acts on data already sorted. If the sort characteristics relate to the different entities who will be billed in respect of the communication instances represented by the data, then this arrangement can also have the advantage that the pricing means can potentially be simplified in applying constraints relevant to individual entities.

It might be noted that a network such as the BT PSTN comprises both local and trunk exchanges and provides not only inter-exchange call transmission but also local call delivery to the end user. This means that the data collection and processing necessary to support billing or bill verification has to be sufficiently complex to deal with an extremely wide range of variables. This is in contrast to the situation where a network provides only inter-trunk exchange transmission, or only the local call delivery.

BRIEF DESCRIPTION OF THE DRAWINGS

A system according to an embodiment of the present invention is now described, by way of example only, with reference to the accompanying drawings, in which:

FIG. **1** shows diagrammatically the architecture of a system for collecting and processing data comprising call information so as to support an accounts system for call instances incoming to a telecommunications network;

FIGS. **2**, **3** and **4** show overview flow diagrams for a system as shown in FIG. **1**;

FIG. **5** shows a hardware and communications diagram for the system of FIG. **1**;

FIG. **6** shows a software architecture for use between a streamer and a data analyzer in a system according to FIG. **1**.

FIG. **7** shows an architecture for hardware providing a company system for use in the system of FIG. **1**;

FIG. **8** shows a schematic diagram of batch array processing architecture for use in the company system of FIG. **7**;

FIGS. **9** and **10** show an exchange file and Advanced Protocol Data Unit (APDU) format, in connection with polling of data from exchanges for use in the system of FIG. **1**;

FIGS. **11** to **21** show flow diagrams for use in a streamer and data analyzer of a system according to FIG. **1**;

FIG. **22** represents process interactions between elements of the system in FIG. **1**;

FIGS. **23** to **30** provide entity life history diagrams, showing the status that a record within each entity might be in, and from that status which other statuses can be reached by which actions;

FIGS. **31** and **32** represent the state of an agenda, and a pattern net, following data population and firing of a rule, in an expert system for use in the system of FIG. **1**;

FIGS. **33** and **34** show object hierarchies, for a rule base system and case base system respectively, for use in a data analyzer of a system according to FIG. **1**;

FIG. **35** shows design principles involved in building an expert system/ORACLE interface for a data analyzer in a system according to FIG. **1**;

FIG. **36** shows a data model for a company system for use in a system according to FIG. **1**;

FIGS. **37** to **43** show flow diagrams relevant to operation of a data analyzer for use in the system of FIG. **1**; and

FIG. **44** shows data a flow, with emphasis on data relevant to a company system for use in a system according to FIG. **1**.

DETAILED DESCRIPTION OF EXEMPLARY EMBODIMENTS

In some parts of the following description and Figures, the terms "INCA" and "IDA" may have been occasionally used. These stand for Inter-Network Call Accounting, a description of the whole system, and for INCA Data Analyzer. The latter is a reference to the data analyzer **7** comprising an expert system and interfacing with the Streamer **6**.

The description below is set out in the following manner:

1. FIG. **1**: BLOCK VIEW OF ARCHITECTURE
2. FIG. **2**, **3**, AND **4**: FLOW DIAGRAMS FOR PROCESS OVERVIEW
   - i) Point of Interconnect and DDC
   - ii) Streamer
   - iii) Company System (or Box)

1. FIG. 1: BLOCK VIEW OF ARCHITECTURE

Referring to FIG. **1**, the system is provided so as to collect data in a first network, for example the BT PSTN, relating to call instances arising in, or incoming from, a second network **2**. The data is collected, at a Point of Interconnect (POI) **3** provided by an exchange of said first network **1**, and brought to one of about ten district data collectors (DDCs) **5** in the PSTN. These hold data which comprises route information for each incoming call, thus allowing identification of for instance the intended destination of the call, the carrier from which the call is incoming, itemization data so that each call is treated as an event, and (preferably) calling line identity so that calls which were simply transit calls in the second network **2** can also be accounted accurately with respect to the network in which they arose.

Each district data collector (DDC) **5** is polled by a streamer system **6** which expands and validates the call data at both file and call record level, primarily against the Routing Reference Model. (Although the district data collectors **5** of the BT PSTN pick up the relevant data, their role may equally be provided by other component systems of an accounting arrangement, such as that known as a network mediation processor.) Data which is found invalid by the Streamer **6** is diverted to a data analyzer **7** where a knowledge-based system is used to access the invalidity and, where possible, reform the data in an attempt to solve the problem. This is an important component of the system since invalid data will generally be lost as an accountable input. Validated data is meanwhile streamed, according to the operator associated with the second network **2** from which the associated call was received, and passed on to the company system **8**.

The streamer **6** provides the following functions:

Poll each DDC **5** for files awaiting processing by the data system of the present invention.

Validate the file and its call records against the Routing Reference Model.

Expand the call records and Allocate to the correct Telecom Network Operator.

Resect the invalid data to the IDA **7**.

Copy the raw file received from the IDA **7** to the Raw Record Backup Interface directory.

Delete the file from DDC **5** once the data has been secured on the interface directory.

Provide the user with a user interface to enter the Routing Reference Model data

Provide a complete audit trail through the streamer.

Provide the user with the ability to monitor the operation and data integrity of the streaming operation.

The data analyzer **7** provides the following functions:

Poll an interface directory for files containing one or more errors.

Hold the incorrect call records in a suspense area if they are valid call records but do not match the Routing Reference Model.

Provide a user interface so that users can re stream the data after the Routing Reference Model has been updated.

Apply default call record values to fields that are incorrect in accordance with the rules specification.

Stream any correct data that has not been streamed already, due to the error thresholds being exceeded.

Stream any corrected data.

Provide a complete audit trail through the IDA **7** at a call record level.

The company system **8** also nowadays has an important role to play because it is the company system which imports factors derived not only from call parameters but also from the relationship between the operators of the two interconnected networks **1**, **2**. The impact a call will have in an accounting procedure will be partly determined by such factors as the "service level agreement" between the relevant operators. It is at the company system **8** that these factors are brought into play, by reference to various information sources which may include look-up tables and/or the National Charging Database (NCDB)**9**. With particular regard to the latter, account is also taken here of for instance time-varying charge rates.

The output from the company system **8** is thus finally information for use in an accounting system, representing the raw call data collected from the point of connection **3**, and processed with reference to the relevant parameters, such as operator-specific and time-varying parameters, which should apply. This output is provided to a client system **10** which gives user access by means of a personal computer.

2. FIGS. 2, 3 AND 4: PROCESS OVERVIEW

Referring to FIGS. **2**, **3** and **4**, flow diagrams can be used to give a process overview of the above, in operation in response to a call instance.

2(i) Point of Interconnect and DDC

FIG. **2** shows process steps carried out by the POI exchange **3** and by the DDC **5** in response to an incoming call. All these steps are known, the exchange **3** and DDC **5** being unmodified for the purpose of the present invention.

Referring to FIG. **2**, a call incoming to or outgoing from the relevant network **1**, at step **200** generates a call record in the POI exchange **3**. At step **210**, the exchange **3** gives every

call instance a "File Generation Number" in the series 0–9999. At step **220**, the exchange **3** groups the call records into Advanced Protocol Data Units (APDUs) and groups the APDUs into files.

At step **230**, the DDC **5** polls the exchange **3** for all call record data in APDU format. At step **235**, the DDC **5** adds control data in the form of a header APDU and a trailer APDU. The DDC **5** also, a step **240**, gives each file a file sequence number in the range from 0–999999, and at step **245** gives each APDU an APDU sequence number in the range 0–16353, APDUs being in binary format. At step **250**, the DDC **5** places the files in a directory structure, from which the Streamer **6** is able to pick them up by polling. At the same time, at step **260**, an entry is made for the file, in a catalogue file which is called DIRINDEX. This catalogue file contains a list of all files available to be polled by the Streamer **6**.

2(ii) Streamer

Referring to FIG. **3**, at step **300**, the Streamer **6** polls the DDC directory structure periodically, entering the call records into a random access memory (RAM), each file being loaded into 1 Mbyte. This polling process includes the step of copying the latest DIRINDEX file. At step **310**, which can be in part of the DDC polling process at step **300**, the data is converted from binary to ASCII (American Standard Code for Information Interchange) format.

At step **320**, the Streamer **6** carries out validation of the call records. If a call record is incomplete or incorrect so that it cannot be verified, instead of proceeding to subsequent processing steps in the Streamer **6** and ultimately to a billing process, it is diverted to an interface directory (step **330**) for further analysis in the incorrect data analyzer **7**.

Valid data however progresses, at step **340**, to an identification process in which data process in which data in the call record is used to establish what other network the call originated in, or entered the BT PSTN from, or in some circumstances was destined to terminate in. A code representing the relevant network operator for billing is added to the call record and the files are then broken down and restructured, at step **350**, according to that code. Hence the call records at this point can be sorted according to the network operator, or other relevant entity, who is liable at least at first instance for the cost of those calls.

At steps **360** and **370**, the Streamer **6** then outputs the newly structured files to the Company System **8** and deletes file data from the FTAM filestore on the DDC **5**.

Looking at the data analyzer **7**, this has an important role to play since data which cannot be validated cannot be billed. The data analyzer **7**, at step **380**, polls the interface directory for errored files entered by the Streamer **6** at step **330**. The data analyzer **7** then has three different chances to put errored data back into the system.

At step **382**, it looks to repair the data. If it can, the repaired data is returned to the interface directory, from which the Streamer **6** can pick it up. At step **384**, the data analyzer **7** looks to apply default values to unrepairable data. Some data elements cannot be "patched" in this manner, for instance because it would affect an audit trail. Lastly, at step **386**, the data analyzer **7** checks whether there is simply a mismatch between the data and the Routing Reference Model (RRM). The latter is a database giving routing information and is used at the DDC **5** to identify for instance the destination of a call. Copies of the RRM are held at different places in a communications network and, if one copy is updated out of time or incorrectly, can give rise to a mismatch in data. If this appears to be the case, the data analyzer **7** enters those call records into a suspend file (step **388**) which allows them go be put back into the Streamer **6** process after the RAM has been checked.

If the data analyzer **7** cannot deal with the data in any of the above ways, it outputs it, at step **390**, to a "sump". This means the data is effectively lost and will never be billed. It might however be useful in analysis so that changes and corrections can be made to the system in the long term.

2(iii) Company System

Referring to FIG. **4**, data, at the file level, which has been validated and processed by the Streamer **6** is input to the Company System **8** where the first step, step **400**, is validation of the file sequence number. The Company System **8** processes files in file sequence number order, but the Streamer **6** has processed data in parallel from different exchanges **3**. If the file sequence number is wrong, the Company System invalidates the file and stops processing it (step **410**).

If the file sequence number is acceptable, the Company System **8** moves on at step **420** to validate the call record, this time not primarily in terms of the RRM, as at the Streamer **6**, but with more emphasis on data relevant to the billable entity and the relationship between the billable entity and the operator of the first network **1**, for instance BT. The billable entity and BT will have entered into a service level agreement (SLA) and the call record might indicate a call type not available to that billable entity under the current SLA. The Company System **8** will pick that up as an invalidity and, at step **430**, attempt to fix the call record in error. If the call record can be fixed, it is sent to be bulked, at step **440**, and re-entered to the Company System **8**. If it cannot be fixed, it is stored, in step **450**, for analysis.

Valid call records meanwhile are forwarded to the Company System pricing engine, step **460**, at which individual call records are priced in accordance with the NCDB **9**, the SLA between the relevant billable entity and BT, and any other relevant information. The priced call records can then be loaded into a summary database, step **470**, for charging to the relevant billable entity, and the call records are output to optical disk (step **480**) for storage.

Client Boxes **10** receive downloaded information from the summary database on a weekly basis. Each Client Box **10** is dedicated to a single billable entity and can also be used to access the optical disk storage, to obtain its "own" call records only.

3. FIGS. 1, 5, 6, 7 AND 8: HARDWARE, COMMUNICATION AND SOFTWARE ARCHITECTURES

3(i) Point of Interconnect **3** and DDC **5**

The exchanges **3** and DDCs **5** are of known type and are not described in detail herein. They operate, briefly, as follows.

Referring to FIGS. **1** and **2**, any call coming into or leaving the British PSTN operated by British Telecommunications plc (BT) will nowadays pass through a digital telephone exchange as the Point of Interconnect (POI)**3**. All such exchanges relevant to the data system of the present invention are currently System X telephone exchanges of types Digital Junction Switching Unit (DJSU), Digital Local Exchange (DLE) or Digital Main Switching Unit (DMSU).

Every telephone call going into or leaving the BT network **1**, as shown at step **200** of FIG. or, generates a call record within the POI exchange **3** in the format known as British Telecom Call Record Type **6**. The System X POI exchanges **3** are polled daily by the DDCs **5**, at step **230**, for all call record data in APDU format. Polling takes place using the File Transfer Access and Management (FTAM) protocol across high-speed BT data links. DDCs **5** act purely as

collectors of call record files from POIs: no processing of call records takes place within a DDC. DDCs are not dedicated to call record polling, but perform a variety of other data collection, processing and forwarding tasks.

In order for the streamer system **6** to gain access to the FTAM filestore on the DDC **5**, it is necessary to provide identification. This is done by allocating a Network Nodal Identity (NNI) to the streamer **6** as a relevant end system. The NNI is then used as a username for gaining access to the FTAM filestore, along with a password.

3(ii) Streamer **6** and Data Analyzer **7**

Referring to FIG. **5**, the hardware and communications diagram for the streamer **6** and the data analyzer **7** may be as follows. (It should be understood that the communications architecture of FIG. **5** represents only one of any number of communications architectures that might be suitable in different environments.) The streamer **6** has a "hot-standby" Streamer Box Backup (SBB) **6**a which cuts in as soon as a fault on the main streamer system **6** occurs, and both can be provided on Hewlett-Packard HP857S minicomputers running the UNIX operating system. The streamer **6** and SBB **6**a might be connected to local area networks (LANs) **515**.

Raw data polled by the streamer **6** (or hot-standby **6**a) from DDCs **5** (not shown in FIG. **5**) is backed up using an optical disc storage system (not shown). The data is polled using FTAM (File Transfer, Access and Management) over BT Megastream high-speed data lines and a Multi-Protocol Routing Network (MPRN) **500**. The MPRN **500** is OSI (Open Systems Interconnection) compliant. There are direct communication links **515** between the streamer **6** and the data analyzer **7** and an "Ethernet" bridge **505** gives the streamer **6** and the data analyzer **7** access to at least one wide area network (WAN) **510**, for instance that used by BT for the PSTN. The WAN **510** in turn gives access to a company system **8** and client boxes **10** situated at the primary BT PSTN network management and data center. This means that the network management and data center can input and output data, for instance for analysis and to input initial and updated routing reference data.

Referring to FIG. **6**, the Data Analyzer **7** might be provided on a Hewlett-Packard HP9000. Software for the Streamer **6** and the Data Analyzer **7** utilizes the following technologies:

IEF for Batch Processes
ART/IM for Expert System Capabilities
HP/UX Version 9.0
Business Objects as a PC Client for reports
Oracle Version 6
SQLFORMS 3
SQL*Report Writer 1.1
PL/SQL Version 1.0
PRO*C
SQL*NET TCP/IP Version 1.2

All these are known and publicly available. For instance "IEF" is the "Information Engineering Facility" Computer Aided Software Engineering (CASE) software from James Martin Associates, a software engineering tool which generates executable code. The data analyzer processes run physically on the data analyzer **7** platform and use SQL*NET to connect to an Oracle database **60** on the Streamer **6** platform. SQL*NET TCP/IP (Transport Control Protocol/Internet Protocol) can also be used by Streamer/Data Analyzer Business Objects Oracle users **65** in order to access the Oracle database **60** located on the Streamer **6** over the MPRN **510**, or a suitable TCP/IP bearer network.

The Streamer **6** and the data analyzer **7** share database facilities **60** and the users may require access for instance to check or update reference data used in validation by the Streamer **6**. The database facilities **60**, inter alia, maintain control over which files from the DDCs **5** have been processed and contain a version of the Routing Reference Model.

PRO*C code is generated by the IEF into the IEF code **61** and External Action Blocks (EABs) **62** as shown in FIG. **6**.

The Streamer/Data Analyzer software library **63** is a set of "C" and PRO*C modules, callable from within EABs **62** or from the ART-IM (Automated Reasoning Tool for Information Management) **64**. The ART-IM is proprietary, expert system, application development software. The ART-IM development is conducted within "studio", a Motif interface to the expert system. Once the expert system has been unit tested within the "studio", it is deployed by generating "C" modules from within the "studio". Hence, for instance, processes can be created by generating the IEF Code **61** on an OS/2 workstation, and linking the code with EABs **62**, the Streamer/Data Analyzer software library **63** and the ART-IM code library **64** on the deployment platform.

3 (iii) Company System **8**

Referring to FIGS. **7** and **8**, the Company Box (or System) **8** comprises a Hewlett-Packard minicomputer **70**, "Emerald 890/400", running the UNIX operating system, the ORACLE relational database management system (RDMS) and a custom application written using the IEF CASE software from James Martin Associates.

Within the Company Box **8**, all call records are priced according to complex pricing and charging reference tables, and ORACLE summary tables are incremented. Reference tables provide exchange set-up data, routing reference data, accounting agreements, pricing and charging data and various classes of exception. Pricing and charging reference tables are derived from BT's National Charging Data Base (NCDB) and inter-operator wholesale pricing agreements.

To help the minicomputer with the very high volume of processing tasks involved, Hewlett-Packard UNIX workstations **80**, for example "735s", are attached as co-processors which bid for processing tasks. A virtually unlimited number of such workstations may be connected to the minicomputer **70** to increase the number of call records that the Company Box can process but a reasonable minimum for the BT PSTN might currently be for instance twelve. As stated earlier, it may be that the data system of the present invention will be required to process 60 million call records per day by 1995. The arrangement relies on the Hewlett Packard product known as "Task Broker" **81**, the data system of the present invention being set up to run on a batch array. In order to do so, custom parameters need to be fed into Task Broker and an appropriate set of these parameters are listed below:

i) Global Parameter Settings (which are optional)
- which clients may access server
- which machines may remotely administer Task Broker
- which network mask to be used
- smallest and largest UID (user identity) allowed
- logging verbosity
- maximum number of task submittals to be processed concurrently.
- list machines that client should contact for services.

ii) Client Parameter Settings (which are optional)
- list for each service, the servers the client should contact for service,

iii) Class Parameter Settings
- every service must be in a class; set up a class for each type of service each machine will provide.

iv) Service Definitions (for every service, the following must be specified)
class
affinity
arguments

Note, affinity is a number between 0–1,000 which indicates how well as node is able to provide a service.

Task Broker is a queuing system which controls which work stations bid for and process files. In order to use Task Broker with the company system **8**, there are three programs and a configuration file. The configuration file sets up the parameters Task Broker needs to operate in the company system environment including which work stations it can communicate with, what programs to call to process a file, and how to prioritize. It is the configuration file parameters which are set out above.

The three control programs operate (in summary) as follows. When a file comes to the Emerald minicomputer of the company system **8**, a master program "run_cp.sh" sends it to be processed via Task Broker and kicks off a monitoring program "cleanup$_{13}$ cp.sh" in the minicomputer. Task Broker allocates the file to a work station, which processes the file according to a third program "cp.sh". If things go smoothly, the file returns to the minicomputer where "cleanup cp.sh" allocates it to the correct directories of a client system **10**. "Cleanup_cp.sh" also monitors the work stations. If there is an over overlong delay in processing by a work station, it will shut down Task Broker on that work station since there is clearly then a problem. Lastly, "cleanup cp.sh" also controls recording and event logging.

Finally, as well as an output to the client system **10**, priced call records from the Company Box **8** are saved to an array of Optical Disc Drives **71**, so That individual priced call records may be retrieved and analyzed in future.

3 (iv) Client System (or Boxes) **10**

Summary ORACLE database tables of interconnect calls are downloaded weekly from the Company Box **8** to Client Boxes **10**. Each Client Box (CLB) **10** is a Hewlett-Packard UNIX workstation, and deals only with summary database tables and call records generated under a single interconnection agreement between BT and another operator, for example Manx Telecom. A Client Box **10** runs in ORACLE RDMS, and Business Objects software. Information from each Client Box **10** allows BT not only to bill another network operator in respect of their use BT's network, but also to verify incoming bills from another network operator to BT. Each Client Box **10** can also interrogate the Optical discs **41**, but only for call records under the interconnection agreement associated with that Client Box **10** it is not possible for a Client Box to interrogate the Company Box **8** directly for its own call records, let alone those relating to other agreements between BT and other operators. Personal Computers are connected to a Client Box **10** to allow analysis of the Summary Tables.

4. FIGS. 9 AND 10: CALL RECORDS AND DATA FORMATS

4 (i) Call Records

British Telecom Type **6** call records are generated for the primary purpose of billing customers. Call records should contain sufficient information to price a call accurately, based on date, time, duration, distance to be travelled and other factors. Each Type **6** call record can include the following:

length of billing record;
record use;
record type;
call type & call effectiveness;
clearing cause;
time discontinuity flag (change to/from GMT from/to BST during call);
calling line identity (CLI);
route group type;
sampling category;
route group;
node point code (NPC): unique identifier for POI exchange producing record;
linking field (used when call straddles more than one time-charge band);
calling party category (business, residential payphone);
charge band;
date and time of address complete;
date and time of answer;
date and time of calling party clear;
date and time of called party clear;
called number field,

Call records are captured by the Call Accounting Subsystem (CAS) Revenue Apportionment and Accounting (RAA) facility on all System X exchanges. As mentioned above, at step **220** call records are grouped together into APDUs, and APDUs are further grouped into a file, with each file being up to 1 Mbyte in size. Nothing in this grouping process within a System X POI exchange destroys any parts of individual call records. All call records are in simple binary format.

Referring to FIG. **9**, each exchange file **40** contains a number of APDUs **51**, which are of variable length. Each APDU **51** contains a number of billing records which are also of a variable length. The following are, however, fixed

Exchange File Maximum Size 1 Megabyte
APDU Maximum Size 512 Bytes
Billing Record Maximum Size 170 Bytes

The DDC Header and Trailer APDUs are identical apart from the APDU type which is 241 for header APDU, and 245 for trailer APDU.

The following information is available in the header and trailer APDU:

APDU Length . . . Length of header/trailer APDU
APDU type . . . 241 for header, 245 for trailer
Unique File Identifier . . . See below concerning DIRINDEX
Destination NNI . . . NNI of INCA Streamer
Application Group . . . Application Group of INCA data=14 Input tape/cartridge
Seq No . . . Sequence Number of tape/cartridge
Output File Seq. No. . . . DDC Sequence Number Timestamp DDC received
data . . . Date and Time data received by DDC
Partfile Indicator . . . Indicates whether file is a part-file
Exception Indicators . . . Indicates what may be wrong with file
Read Count . . . No. of times this file has been read
Filesize . . . Size in bytes of this file Count of unselected
APDUs . . . No. of APDUs of wrong APDU type
Selected APDU type . . . APDU type of INCA data type
APDU Count . . . Number of APDUs in this file
First Seq. No . . . Starting APDU Sequence Number
Last Seq. No . . . Ending APDU Sequence Number

The read count represents the number of times this file has been polled from the DDC by the Streamer **6**. The partfile indicator indicates whether the whole file was received by the DDC **5** successfully or whether parts of the file were missing.

The exception indicator are two 1 byte bitmask fields which indicate any errors that were detected by the DDC **5** relating to this transfer.

The valid values for all of the fields above will be validated within the computer aided software engineering (CASE) application software described below with reference to the "COMPANY SYSTEM (OR BOX)" **8**.

Referring to FIG. **10**, a brief description of the APDU structure **51** would indicate the APDU header **52**, the actual billing records **53** concerned, and the APDU trailer **54**.

The format for the billing records **53** is of standard type and a "C" structure can be designed to map exactly onto that format.

When data has been polled from the exchanges **3** to the DDC **5**, some of the data which is at the head of each data APDU is stripped out by the DDC **5**. This data is representative of the DDC **5** and of the exchange **3** and is not relevant as a data feed for an end-processing system.

When the file is copied into an appropriate directory by a DDC **5**, such that it is made available for the streamer **6** to copy using FTAM, an entry is made for it in a catalogue file, called DIRINDEX. The DIRINDEX file entry carries the following data:

i) activity marker (1 byte) which may show
  a) inactive entry
  b) file available for transfer
  c) file currently being used (eg in FTAM transfer)
  d) file successfully transferred (not yet deleted)

ii) INCA filename format

iii) output routine, which may show
  a) file available for FTAM
  b) magnetic tape only

iv) unique file identifier, including details such as the creation time and the relevant exchange NNI.

v) file size in bytes

vi) number of APDUs in file.

Looking at ii), the INCA filename format, that includes:

vii) the streamer NNI

viii) NNI and cluster number of exchange

ix) application group of INCA data

x) DDC file sequence number of exchange file.

4 (ii) Mapping Data Structures onto Exchange Data

The streamer **6** maps the data into data structures for use in the model for the Company Box **8**, using the following principles:

It is assumed that the APDU length fields and the billing record length fields will be correct. If they are not then the validation will fail at either the APDU level or the Billing Record level, and the file will be streamed to the Data Analyzer **7**.

The input data will initially be scanned to find the Header APDU **52**. This will be identified by an APDU type of 241 (Hex F1). The selected APDU type field will then be checked alone with the Unique File Identifier to establish that this is indeed the header APDU **52**.

After the header **52** has been found and the header APDU data structure has been mapped, it is assumed that all of the APDUs in the file will follow the data standard of a one word record length followed by an APDU. eg./HEADER_APDU/ RL/APDU/RL/APDU . . . /RL/APDU/RL/TRAILER_ where RL is the Record Length.

If the structure of the file deviates from this standard then the file will be streamed to the Data Analyzer **7** for further analysis. This error condition will be detected within the validation of the APDU immediately following the deviation.

Within each APDU it is assumed that the structure follows that of FIG. **6**. Again any deviation from this structure will cause the whole data structure mapping to become misaligned, and will lead to the file being rejected and streamed to the Data Analyzer **7**.

It is assumed that there will be no data after the trailer APDU **54**. Any data that does appear after the trailer APDU **54** will be lost.

5. FIGS. 11 TO 19 AND 22 TO 30: STREAMER AND DATA ANALYZER PROCESSES

5 (i) Streamer: DDC Polling Process

When files are received by the DDCs **5** they are validated (using checksumming) and some extra information is added to the beginning and end of the file, in the APDU Header and Trailer **52**, **54**, as mentioned above with reference to FIG. **2**. These files are then made available for polling by the Streamer **6** by placing them in the directory structure to be used by the streamer **6**, and updating the DIRINDEX file. This DIRINDEX file contains a list of all files available to be polled by the Streamer **6**, and the Streamer **6** uses that list to ensure it has polled all new files.

Referring to FIG. **11**, the Streamer **6** will prepare to poll multiple DDCs **5** to going into a "Stream all DDCs" process. At step **700**, the streamer **6** "stream all DDCs" process is triggered, for instance at a specified time. At step **710**, it runs a check that the Streamer **6** is available to receive files from the DDCs **5**. If the Streamer **6** is available, it goes into a cycle, steps **720**, **730** and **740**, in which it runs through a list of the DDCs **5** and creates a "DDC Process" for each DDC **5** to be polled. At the end of the list, this process finishes (step **750**).

For each DDC **5**, the Streamer **6** will now run the "DDC process". Referring to FIG. **12**, at steps **800**, **805**, the DDC process starts with a check as to whether either the DDC **5** concerned for that process, or the Streamer **6**, is shut down, and a check at step **810** as to whether DDC polling is due. There are certain times at which the DDCs **5** cannot be polled and step **815** runs a check as to whether a non-poll window applies. If not, step **820** looks for a free process slot to process files. If all these checks are clear, the streamer **6** accesses the DDC DIRINDEX, step **825**, and initiates the process list, step **830**, and file list, step **835**, which will ensure the streamer **6** applies all relevant processes to each of the exchange files received from the DDC **5**. In step **840**, **845** and **850**, the streamer **6** runs through the files from the DDC DIRINDEX, creating its own log of the exchange files to be processed, and provides a file processing capacity, steps **855** and **860**, to process the files in the file list. Once all the exchange files from the DDC DIRINDEX list have had processing capacity allocated, the "DDC process" updates its record of when the next poll is due, step **865**, and goes back to sleep, step **870**.

The DOC process will be stopped of course, step **875**, if either the DDC **5** or the streamer **6** has shut down, and will remain in sleep mode, step **870**, whenever a poll is not due, the DDC is in a non-poll window, or there is no available processing capacity.

typical event cycle within ddc polling architecture

Assume the following

DDC_POLLING_MPH=17 . . . This is the minutes past the hour to Poll

DDC_POLLING_INT_HRS=1 . . . This is how long to wait for next Poll in hours

DDC_DELAY_IN_DELETE=12 . . . How long to wait after the file has been marked for deletion before actual deletion request occurs.

The System has been booted at 23:30 on the previous day.

Time Schedule	
00:17	DDC Process wakes up, copies over DIRINDEX file, and creates processes to stream data to the Streamer 6.
00:30	DDC Process finishes creating processes to stream files because either the Maximum number of processes have been created OR all of the files available have been given to file processes to download. The wakeuptime is calculated as 00:30 + DDC_ POLLING_INT_HRS and set minutes to DDC_POLLING_MPH. => Next Polling time + 00:30 + 1:00 = 1:30 (SET MPH) = 01:17 Calculate the number of seconds to sleep = TO_ SECONDS (01:17 - CURRENT_TIME) Sleep (seconds_to_sleep) . . . File Processes complete streaming of data
01:17	DDC Process Wakes up . . .

5 (ii) Streamer: File Process

Referring to FIG. **13**, the operation of the File Process, created at step **855** during the DDC Process, is as follows. The File Process works from a file list received from the DDC Process, step **1302**. Running through the file list, step **1303**, for each exchange file listed, File Process reads the exchange file log, step **1305**, validates the call records, step **1306**, copies the file to a raw record backup, step **1307**, for use if for instance the streamer **6** goes down subsequently, diverts the file to the data analyzer **7** if there was a validation failure, step **1310**, or streams the file to the Company Box **8**, step **1312**.

File Process stops, step **1313**, if the DDC **5** or the Streamer **6** is shut down, step **1304** or if the files are seriously corrupted, step **1311**, for instance because communications with the DDC **5** have failed. The exchange file log monitors what stage an exchange file has reached in relation to the Streamer **6** and will carry a status selected from active, processed and deleted for each file, where "active" indicates it is being processed by the Streamer **6**, "processed" indicates it has been processed by the Streamer **6**, and "deleted" means it has been deleted from the DDC **5** by the Streamer **6**.

Referring to FIG. **14**, the step **1306** in which call records are validated can be expanded as follows. At this point, steps **1401** and **1042**, the exchange file is copied from the DDC **5** and the file header and first APDU header **52** validated, steps **1403**, **1405**. If either fails, a file error log is created, step **1412**. If both are acceptable, the call records are each validated steps **1407**, **1408**, and a call record error log created, step **1409**, if one fails. Validation is repeated for each APDU **51**. Whether validation has shown all is correct, or errors have been logged, the audit trail is updated **1413** and File Process moves on to step **1307** as described above.

Referring to FIG. **15**, files which have been validated during the File Process are now ready to go to the Company Box **8**. At this stage, the file structures are broken down so that the individual call records **53** can be sorted according to the billable entity they are relevant to. The call records **53** are now written to physical files, step **1503**, for the different billable entities.

5 (iii) DDC File Deletion Process

Once a data file has been successfully downloaded from the DDC **5** to the streamer **6**, and the data has been expanded and streamed to the appropriate Company Box **8**, the data file must be deleted from the FTAM filestore on the DDC. The streamer **6** will delete the file using an FTAM delete request a number of hours after the file has been secured on either the company box **8** (or local storage for the company box **8** if the link to the company box **8** has gone down). The exact time between the data being secured and the files being deleted can be set on a per DDC basis.

5 (iv) Data Analyzer Process

Referring to FIG. **16**, the step of validating call records in FILE PROCESS, step **1306** in FIG. **13**, generates a file error log, step **1412**, and a call record error log, step **1409**. The data analyzer **7** runs two processes, the "DA PROCESS" and the "SUSPENSE FILE PROCESS", which are initiated during the boot-up sequence of the HP9000.

DAPROCESS monitors continuously whether data which has been sent by the Streamer **6** is available to be processed by the Data Analyzer **7** This data will always exist initially as the original exchange file, irrespective of whether the data contained individual call records which could not be steamed, or the failure was at file level.

As long as the Data Analyzer **7** is not flagged as shut down, step **1602**, DA PROCESS will first pick up the earliest file error log to be Processed, step **1603**, and check whether it was a failure at file APDU level or at call record level, step **1604**.

Referring to FIGS. **16**, **17** and **20**, if the failure was at call record level, DA PROCESS will pick up the next call record error log with regard to the file, step **1702**, and send the relevant call record to the ART IM rule base for correction, step **2000**. If the failure was at file level, the whole exchange file has been rejected by the Streamer **6**. In this case, the complete file is loaded to memory, step **1606**, and the file header and APDUs **51** sent to the ART IM, step **1607**, for correction.

There are several outcomes to analysis done by the Data Analyzer **7**. Fixable data will be sent to the ART IM to be corrected, and subsequently can be validated and streamed to the Company Box **8**. If a routing error is involved, the data may be put into suspense in case there is a problem with a record of routing information somewhere in the system, for instance because it needs updating. It may be possible to validate call data after all, once the routing information has been corrected. If a whole file is unreadable, it might have to be sent, still in binary format, to a Binary File Dump. If data, for instance a file, is determined by the ART IM to be unfixable, and the error is not concerned with routing so as to justify suspension, it may be archived. The data will never be billed but may be used in analysis to identify long term or significant problems which themselves can be put right and so avoid losing billable items in the future.

Returning to FIG. **16**, the main DA PROCESS, having used the ART IM to run checks at step **1605** and **1607**, will next sort out files which have been returned from the ART IM as unfixable. If they cannot even be read, step **1608**, they are forwarded to the binary file dump. These files can potentially be read, since they may be in hexadecimal, octal or ASCII format, and might be used at a later time for analysis. Alternatively, files might be readable by the Data Analyzer, but are still rated "unfixable" by the ART IM. These are, at step **1609**, loaded to a "SUMP" database where, again, they will never provide billable data but can be queried and analysed.

If a file has been sent to the ART IM and was fixable, the ART IM will return each call record sequentially for validation, step **1610** and **1611**. DA PROCESS will then validate these call records first by checking for a routing

failure, step **161**, and creating a call record error log, step **1615**, in the event that there is call record failure. These will get picked up and return through the ART IM, steps **1603** to **1605** and **1701** to **1703**. If the call record is acceptable, it will be streamed to the Company Box **8**, via steps **161** to **1618**.

Referring to FIG. **18**, where there has been a call record routing failure, detected at steps **1612**, **1704**, or **1907** (see below), the call records are categorised and suspended. That is, the failure is analyzed to the extent that it can be matched to an existing routing error pattern, step **1802**, and then the call record is added to an existing pattern file which contains all call records showing the same routing error pattern, step **1803**. These pattern files are held in suspension, the primary BT PSTN network management and data center being notified. A separate process, SUSPENSE FILE PROCESS, then deals with these files.

SUSPENSE FILE PROCESS is an important aspect of the data analyzer **7** because it takes a category of errored files, which can potentially be corrected, out of the "mainstream" of data processing. These files may only have been picked up as errored because routing data somewhere in the system has not been updated. They are potentially billable. By means of SUSPENSE FILE PROCESS, the primary network management and data center has the opportunity to update routing data in the system and still catch files found errored previously. Further, by appending call records to an existing pattern file, a "Route Pattern Suspend File", for a particular route pattern, files can be selected for reattempting validation by simply running a selected Route Pattern Suspend File.

Referring to FIG. **19**, as long as the process is not shut down, step **1902**, SUSPENSE FILE PROCESS starts by locating the earliest routing pattern which has been amended, for instance, by the network management and data center, step **1903**. It will then pick up the next suspended file containing that routing pattern, step **1904**, and attempt to validate the call records, steps **1905** and **1906**. There may of course be more than one routing error in the call record. If that is the case, SUSPENSE FILE PROCESS will revert to step **1801**, on FIG. **18**, and create a routing error entry in a routing error pattern file, thus re-suspending the call record. However, if there is no other routing failure, SUSPENSE FILE PROCESS will attempt to stream the call record to the Company Box **8**, by reverting to step **1501** on FIG. **15**. The PROCESS runs through all of the call records in the suspended file in this way, step **1910**, and all the files which have been suspended with respect to that particular route pattern, step **1911**.

Referring to FIG. **22**, this shows the process interactions between the streamer system **6**, the company box **8** and the data analyzer **7**. The main process area of the streamer **6** is the one called "FILEPROCESS". This does all the validation and intrinsic operations on a file. In the data analyzer area there is the "IDA FILEPROCESS" which enters data to the expert system. Importantly, this process triggers the Route Pattern Suspend File and "SUSPENSE FILEPROCESS" by appending data to a Route Pattern Suspend File. It is this which avoids a large backlog of data building up because SUSPENSE FILEPROCESS operates outside the main IDA FILEPROCESS. Another area of interest is the "SUMPDATABASE" receiving output from the "SUMPLOADER". Although data in the SUMPDATABASE cannot be put right, it can be queried and analyzed so that, potentially, rules at the IDA FILEPROCESS can be changed so that subsequent data can be re-streamed.

In FIG. **22**, processes are shown in circles, the Company Box **8** as a block, data files, logs and the like are shown between open-ended parallel lines and archived data is represented by the conventional symbols for databases.

This process, and stored data, interactions referenced (a) to (y) on FIG. **22** can be listed as follows, the arrow heads denoting relevant transfer directions:

a) NNI and list of file names to be processed, transferred
b) Exchange file log, STATUS=A, created
c) DIRINDEX file accessed
d) FTAM exchange file copied
e) FTAM exchange file deleted
f) Exchange files, where status=P, read
g) STATUS set to D if exchange files deleted successfully (at (e) above)
h) Exchange file log read where status=A
i) Exchange file log data updated. STATUS set to P
j) File is in error so file error log created
k) Call record is in error so call record error log created
l) File copied to Data Analyzer directory if file is in error
m) File error log read
n) Call record error log read
o) Raw (binary) data file looked up
p) Data appended to route pattern suspend file for this route pattern
q) Entry made in route error pattern
r) ART/IM created closet matches
s) ART/IM has identified that this data cannot be fixed. Data is placed in the SUMP for further analysis or deletion
t) User has identified the problem cannot be fixed. File is placed into the sump for further analysis or deletion
u) When file structure unintelligible, file thrown into binary file dumps
v) Streamed file created.
w) SUSPEND FILE PROCESS is initiated by status on route error pattern being set to ready. If problems persist then Count field updated and status set to SUSPENDED
x) Closest matches are updated if the chosen solution fails to fix the problem
y) Streamed file created

(v) Entity Life Histories

Referring to FIGS. **23** to **30**, entity life history diagrams can show the statuses that a record within the entity can be in and, from that state, which other states can be reached by which actions. In each of these Figures, the statuses are simply identified by the reference numeral **2300** and the definitions of the statuses are given below.

FIGS. **23**: File Error Log;

READY—the file is ready to be streamed by the data analyzer **7**.

SUSPENSE—either the whole file or a least one call record within the file has been sent to the suspense area.

BIN—the file could not be read by the data analyzer **7** and has been sent to the bin area.

SUMP—the whole file has been sent to the sump area.

COMPLETE—the data analyzer **7** has streamed the file and any of the files call records in the suspense area have been successfully re-streamed or archived.

FIG. **24**: Call record error log;

READY—call record is ready to be streamed by the data analyzer **7**.

SUSPENSE—call record has been sent to the suspense area.

SUMP—call record has been sent to the sump area.

ARCHIVED—call record has been sent to the trash area (ie ARCHIVED).

COMPLETE—the data analyzer **7** has streamed the call record successfully.

VAL_FAILURE—there are differences in the ART-IM and IEF validation procedures.

FIG. **25**: Route error pattern;

UNSELECTED—created by ART-IM and waiting analysis by data analyzer user, or re-streamed after analysis but failed.

PENDING—selected by data analyzer user for analysis.

READING—data analyzer user completed analysis and is ready to be re-streamed.

CLOSED—successfully re-streamed or ARCHIVED.

FIG. **26**: Closest matches:

UNSELECTED (OR NULL)—generated by ART-IM

SELECTED—selected by data analyzer user for analysis.

FIG. **27**: Sump file log

SUMP—a file is ready for the SUMP_PROCESS.

PROCESSING—the file is ready to be viewed by the data analyzer user.

ARCHIVED—the file has been archived.

FIGS. **28**: File route error link;

SUSPENDED—the file is in the suspense area.

COMPLETE—the file has been successfully re-streamed from the suspense area.

FIG. **29**: Exchange file log;

A(CTIVE)—exchange file is being processed by the streamer.

P(ROCESSED)—exchange file has been processed by the Streamer.

D(ELETED)—exchange file has been detected by the Streamer

FIG. **30**: District data collector;

(All statuses are changed by Steamer **6** users via SQL*Forms.)

P(REBIS)—DDC is prebis.

L(IVE)—DDC is live.

C(EASED)—DDC has been ceased.

Referring to FIG. **6**, it will be sent that the Streamer **6**/Data Analyzer **7** software architecture includes IEF external action blocks (EABs) **62**. The EABs **62** are used where it is inappropriate or not possible to implement within the IEF. For instance, the following functions might be carried out by means of the EABs **62**:

"Add call record to suspense"

This module will create a new entry of a call record, within a linked list containing call records for an exchange file which are to be set to the suspense file.

"Add call record to archive".

Creates a new entry of a call record, within a linked list containing call records for an exchange file which have been fixed but cannot be re-streamed, to the archive directory.

"Add network operator record".

Checks whether this is a new Network Operator and if so will create a new entry in the linked list of "network_operator_structure". If it is an already used network operator name it will add a linked list entry into the linked list of call records for that network operator. Where a fix has been applied. So a call record it will update the "call_record_error_log" with the "network operator record" identity and streamed call record sequence number.

"Call record to IDA rules"

Passes a single call record to the data analyzer ART-IM rule base. The call record is identified by the APDU sequence number and call record sequence number passed in by IEF. The data structure loaded in memory is then searched for the call record and owning APDU and exchange file header data. This data is then fed into the rule base and validated. Any errors found will each generate a call record rule log row entry. The call record error log record status will also be updated by the rule base.

"Commit"

Commits all current database changes.

"Create DDC process"

Creates an occurrence of the DDC process which will be responsible for polling a particular DDC. It will create/open a fifo (file in/file out) to the child process and will write into the fifo the value of the DDC_NNI.

"Create file process"

Creates the process which will perform the task of streaming the file names passed in the array of file names.

"Delete file from DDC"

Deletes a file using the FTAM protocol from disc on the DDC.

"Delete data analyzer file"

Deletes a file from the streamer/data analyzer directory.

"Delete suspense file"

Deletes a file from the suspense file directory.

"File to bin"

Passes a file which cannot be read into the ART-IM rule base to the binary file dump.

"File to data analyzer rules"

Passes a whole file to the data analyzer ART-IM rule base and initializes the rule base. The initialisation of the rule base involves clearing odd data, selecting default and routing reference data and populating the rule base with that data. The data analyzer binary file is then loaded into a data structure in memory. This data is then fed into the rule base and validated. Any errors found will each generate the appropriate rule log row entry. Call record error logs will be created by the rule base where appropriate, together with route error pattern, closest matches and file route error link records. Once validated, the rule base will return a validation status to IEF and retain its internal data for later retrieval.

"File to sump"

Passes a file which cannot be fixed by the ART-IM rule base to the sump.

"FTAM HLCOPY"

Copies, using the FTAM protocol, the DDC file name from the DDC FTAM address using the DDC user name, DDC password and DDC account to the Streamer **6**. The user name, password, account and FTAM address of the Streamer **6** can be defaulted to NULL if required. The routine is not called directly from the IEF and hence does not return an IEF style status.

"Get DDC process parameters"

Creates or opens a fifo in the streamer/TMP directory which will be named "DDC_process_fifo<PID>. It will read the values of the DDC NNI from the fifo, the data having been inserted into the fifo by the "create_file_process" routine.

"Get file process parameters"

Creates or opens a fife in the streamer/TMP directory which will be named "file_process_fifo_<PID>". It will read the values of the above variables from the fifo, the data having been inserted into the fifo by the "create_file_process" routine.

"Get exchange file"

Copies a file using the FTAM protocol from disc on the DDC, straight onto disc on the Streamer **6**. The file will then be read into memory on the Streamer **6** and then renamed into the raw record backup directory from where it will be archived. This module calls "map_data_structure_to_file" in order to set up the initial pointers to the first billing record, first APDU, and the header and trailer APDUs.

"Get no. invalid data analyzer APDUs"

Returns a count of invalid APDUs re-processing of call records which have failed the Streamer validation process.

"Map data analyzer file"

Reads a file into memory for subsequent processing.

"Process active"

Establishes whether a particular PID is active and returns a flag accordingly.

"Read exchange file header"

Uses the pointers to the header and trailer APDU to return in a structure all of the fields from the header and the APDU type from the trailer.

"Read data analyzer exchange file header"

Uses the pointers to the header and trailer APDU to return in a structure all of the fields from the header and the APDU type from the trailer for a file which has been sent to the data analyzer.

"Read first DIRINDEX record"

Copies, using the FTAM protocol, the DIRINDEX file from the DDC to temporary storage, and opens the file and returns the first record to the caller.

"Read next APDU"

Returns the APDU structure pointed to by the current APDU pointer and sets the current APDU pointer to the next APDU. Also sets the current billing record pointer to the first billing record within the returned APDU, and copies and byte reverses the data into the current APDU array.

"READ next DIRINDEX records"

Reads the next record from the DIRINDEX file on the DDC.

"Read next data analyzer record"

Returns the next billing record output from the ART-IM rule base. Successfully processed records will appear first, followed by those which require sending to the suspense file.

"Read next suspense record"

Returns the next billing record output from the suspense file.

"Read next record"

Returns the billing record currently pointed to by the current billing record pointer, and sets the point to the next billing record if this record is not in the last in the current APDU. (This is determined using the APDU length and the minimum length of a billing record.)

"Rename network operator files"

Renames any network operator files that have been written to the temporary directory in the operational directory ready to be processed by the Company Box **8**.

"Sleep"

Will sleep the specified number of seconds.

"Stream file"

Dumps the file in memory to the data analyzer ready for data analyzer processing.

"Stream file network operator"

Uses the pointer to the first network operator to get to all of the validated, expanded records for that operator. It then attempts to write the records from the linked list into a nfs_temporary directory. If successful, the file is renamed into the nfs_directory. If the file cannot be reopened on the nfs temporary directory, the file is opened on the local temporary directory, and upon successful writing the file is renamed into the local directory.

"Stream file RRB"

Dumps the file in memory to the raw record backup directory.

"Write console"

Writes a message to a network management workstation.

Write to suspend file"

Writes the records from the linked list of suspended call records for an exchange file into the suspend directory.

"Write to archive file"

Writes the records from the linked list of archive call records for an exchange file into the archive directory.

6. FIGS. 31 TO 35: EXPERT SYSTEM; ART-IM

6 (i) Overview

The expert system uses the facilities of the ART-IM knowledge based expert system tool kit supplied by Inference Corporation. It is a knowledge/rule base programming system which allows for a flexible model of decision making, and therefore modelling of the real world, within the knowledge hierarchy, as well as providing a more heuristic method for problem solving. The tool kit contains the ART-IM language as well as an integrated editor, and interactive development environment, tools for the development of end-user interfaces, a method of deploying run time versions of developed applications and the facility to interpret external data intelligently.

In the data analyzer **7**, the expert system is split into two subsystems, the rule base and the case base. In general, the case base is used to deal with routing based errors, and the rule base is used for defaulting and calculable errors. Both make use of the ART-IM functionality.

The rule base uses the ART-IM procedural languages including rule, function and methods. Each error is defined within a schema and instances of these schemes are used on the data structures. All schemes within an in-data object hierarchy are populated via the IEF/ART-IM interface using the "DEF-EXTERNAL_FUN" facility of ART-IM.

The mechanism of program flow control used by ART-IM is very different from sequential statement-by-statement flow, as usually found in programming languages. Referring to FIGS. **31** and **32**, the expert system holds all its internal data, that is schemata and facts, in a pattern net **3100**. This is represented in FIG. **31** by a plurality of patterned circles, each representing a piece of internal data (a schema or a fact). This data can be set up by

- loading an ART-IM test case file (more usually done in a development/unit testing context).
- by populating from an external source (eg Oracle or IEF; more usual in a production/system test environment).
- by generating from ART-IM rules (used as very flexible "working storage" eg generation of error schema after validation test failure).

Once set up, data is compared directly with the conditions specified within the rules. A rule resembles an "IF<conditions>THEN<action>" of a more traditional programming language. If conditions of the rule match exactly an instance of data, an activation is created within an associated agenda **3105**. All instances are checked against all rules. In terms of performance, the pattern net and rule conditions are managed by an efficient pattern-matching algorithm within the ART-IM run time system.

At the end of the evaluation part of the cycle, all rule activations are placed in order on the agenda stack. The first rule activation on the stack will be fired. The order of appearance of activations defaults to random unless salience, that is priority of rules, is set by the developer.

Referring to FIG. **32**, after firing of the topmost rule activation on the agenda **3105**, the action of the rule has actually changed data in the pattern net which will in turn after what appears on the agenda stack following the next evaluation cycle.

It might be noted that the data instance causing the initial firing (the circled instance **3110**) will not be reevaluated, thereby avoiding continuous looping although if the data within the data instance changes and the new pattern matches a rule condition, then a rule activation will be created.

The ART-IM run will finish when:

no matching conditions and patterns found. all matching conditions and patterns have already fired rules.

The above can be summarized as follows:

1) rule activations are generated by matching data patterns with rule conditions

2) rules can, by default, fire in any order although priorities can be set

3) all data is evaluated in parallel

4) re-evaluation occurs each time a rule has fired

5) the same rule can fire many times during a run, depending on the number of matching data instances

6) rule conditions are sensitive to changes in the pattern net

7) ART-IM stops if no matching rule conditions or pattern net data is found or all matched activations have fired already.

Referring to FIG. **33**, the rule base system is based on an object hierarchy as shown. Each of the objects **3300** is defined in ART schemes and she connecting lines between objects **3300** depict inheritance from the object above.

The exchange file, APDU and call record contain slots for each data item in their structure. Each slot has a corresponding slot in the appropriate default object to declare whether the resultant has a default value, a calculable value or is an un-modified field. The rule base uses the default system to check what form an error correction must be, if allowed.

The above covers the data schemas. With regard to error schemas, every possible data analyzer error has its details described within an appropriate schema. Each error description and its instances contains a slot for each of the following:

The object on which the error relates, that is an exchange file.

An error description.

The affected slot.

The specific data object for an error instance.

The name of the repair value.

The source of the error.

The resultant repair value.

The rule position in fire order.

The value of the slot prior to any fix being applied.

6 (ii) Rule Base Generic Rules

The rule base operational flow is controlled by a number of generic rules, these performing the following functions:

for each occurrence of an error trigger, that error's repair method is fired to generate a repair value and its fire order allocated.

for a fixable error where there is only one affected slot, the affected slot is updated with the repair value generated and the time stamp of the change is stored with the instance of the error.

for each instance of an error where the repair description declares the error type as suspendable, the data item affected is moved to the suspense file and the time stamp of the move is store with the instance of the error.

for each instance of an error where the repair of description declares that the error type is sumpable, the data item affected is moved to the sump and the time stamp of the sumping of the file is stored with the instance of the error.

a record is created on the file structure rule log for each fix on an APDU or exchange file.

an Oracle record is created on the call record error log for each fix on a call record with the appropriate error information.

fixable errors

1) Default values can be allocated to the following fields:

APDU type and trailer

billed call indicator

called party clear

PBX suffix

record use

record type

DDC time stamp

header APDU type

class of data transfer

format version number

node time stamp

part file indicator

table size

trailer APDU type

called party clear

application

2) The following errors are calculable:

APDU length; the length of the APDU.

APDU count; the length of the APDU sequence.

End APDU sequence number; start sequence number plus the number of valid APDUs.

Start APDU sequence number; obtained from the sequence member of the first APDU in the exchange file.

Dialed digit count; the length of the dialled digit string.

There are error exceptions with regard to the above, such as where the checksumming for an APDU shows an error. Errors of this type are immediately sumped within the rule base. Some errors with regard to the APDU sequence result in the complete range of sequence numbers being re-sequenced from "1", and the related exchange files being updated. It may be that the last digit of a dialled digit string is a character between A and F. The repair value here is the dialled digit string minus the last digit.

non-fixable errors

On the non-fixable error occurrence, the data item in error, ie a call record, is passed to the sump, as described above, and the appropriate error log updated. Areas which cannot be amended, and which therefore generate non-fixable errors are as follows:

address seizure time stamp

address completion time stamp

either address or answer time stamp

calling party clear time stamp

calling line directory number

seizure time stamp

dialled digit string (except when the last digit is between A and F)

6 (iii) The Case Base System

The routing reference case base is a case base of routing patterns (ie TUN, route group, route number, NNI, Nodal Point Code) plus other reference data, for example the billable network operator name, and Live and Ceased Node time stamps. The case base reference data is populated from the Routing Reference schemata which in turn are populated from data contained within the Streamer Reference Data subject area **3600** of the data model (See FIG. **36**).

Referring to FIG. **34**, it can be seen that the object hierarchy for the case base system is similar to that for the rule base system, shown in FIG. **33**, with the addition of three object classes **3400**; "suggested solutions", "potential solutions" and "routing reference". It might be noted that "suggested solutions" and "potential solutions" are only created following identification of a routing reference error and contain mainly pointers to other data, that is incoming call record in error and routing reference schema that are most closely matched. The "routing reference" schemata are created from the routing reference data on the Oracle database.

With regard to the routing case base, and initialization, the routing case base is populated at the start of a run from routing-description schemata. One case is created for every routing description schema. The routing case base will be set up with the following parameters.

Maximum of three matches

Any matches below threshold value of zero probability will be ignored so as to weed out highly unlikely matches.

Only the following slots on the case base are used in pattern including:

TUN (ie Telephony Unit Number), route group, nodal point code, route number, NNI, and direction

The following are ignored for purposes of pattern matching:

Live node Lime stamp

Ceased node time stamp

Telecom network operator role and name

Direction is treated as slightly different for matching purposes. It is the least significant matching slot and is given a fixed weighting ceiling of 5% of the overall weighing. The other slot weights will be split equally between the remaining 95% of the overall weighing.

Pattern matching, together with other case base functions such as setting initialization parameters, is achieved by sending messages to the case base. The pattern matching is done in two steps, these being to send an incoming call record schema to the case base, which will return the number of matches found, and to send a retrieve-match-score message which will determine the closeness of match for each returned case together with the key of the Routing Reference schema associated with the returned case.

The case base is used for an error code validation relating to call pattern, in the cases of Nodal Point Code or Route Group Not Found, Invalid Route Number, or Direction Invalid, as follows:

attempt to find an exact match between each incoming call record and a case on the Routing Reference case base. If there is an exact match the call record has a valid routing pattern and no further validation with regard to the above error will be required.

if no exact match if found, an error schema will be generated which triggers the rule base generic rules, as above, which will apply a repair method.

the specific repair method will create one suggested-solution schema which will contain (looking at FIG. **34**):

i) up to three potential-solution schemata, each containing a pointer to the associated Routing Reference schema. The potential-solution schema will also contain the match position (ie closest, next closest etc) and a % measure of the closeness of the match, and

ii) a pointer to the incoming call record in error.

It should he noted that the repair method will differ from the usual repair method invoked by generation of an error schema instance because it will consist of a function (routing-mismatch, which will assert the suggested-solution schema instance and facts containing keys to the Routing Reference schema) and another rule (generate-closest-matches, which will trigger on generation of the facts created by routing-mismatch and will generate one instance of a potential-solution schema for each case base match found).

Where node time stamp validation is concerned, the case base will be used as follows:

to attempt to find an exact match between each incoming call record schema and a Routing Reference schema. If there is an exact match the rule will then check for time stamp discrepancies (ie seizure time stamp should fall between node live and cease times) on the matching incoming call record schema and the Routing Reference schema. If no discrepancy exists, no further processing associated with this error will take place.

if a time stamp discrepancy is found, an error schema will be generated which triggers the rule base generic rules, as above, which will apply a repair method.

the specific repair method will create one suggested-solution schema will contain (see FIG. **34**):

one potential-solution schemata each containing a pointer to the associated Routing Reference schema. The potential-solution schema will also contain the match position (ie closest, next closest etc.) and a % measure of the closeness of the match.

a pointer to the incoming call record schema in error.

It should be noted that the repair method will again differ from the usual repair method invoked by generation of an error schema instance, because it will consist of a function (node-timestamp-discrepancy—which will assert the suggested-solution schema instance and facts containing keys to the Routing Reference schema) and another rule (generate_node_time_discrepancies which will trigger on generation of the facts created by routing-mismatch and will generate one instance of a potential-solution schema).

6 (iv) ART-IM and Oracle Interface

Referring to FIG. **35**, direct access to the ORACLE database from ART-IM is required to exploit fully the "Parallel" validation features of the ART-IM rule base. There are four main interfaces:

the population of Routing Reference data **3500**

the population of default data **3505**

the output of fix data to form an audit trail **3510**

the output of routing error patterns as a recursor to suspense data handling **3515**.

Looking at the population of Routing Reference data, this interface **3500** involves refresh of internal ART-IM schema and casebase from data in the Routing Reference Model physically held within ORACLE tables:

the refresh is triggered during the initialization phase of an ART-IM run.

existing internal ART-IM Routing Reference schema are cleared together with their casebase entries.

data is SELECTED from ORACLE tables from a ProC program (EAB_INITIALIZE_IDA_RULEBASE)

which will be used as part of two External Action Blocks (file_to_ida_rules and call_record_to_ida_rules).

Internal ART-IM schema are populated by the ProC program

The Routing Reference Casebase is in turn populated from the internal Routing Reference schema by a function (inca_ida_initialize_casebase) as part of the casebase initialization process.

Looking at the population of default data:

the refresh is triggered during the initialisation phase of an ART-IM run.

existing internal ART-IM default (df-call-record, df-apdu etc) schemata are cleared together with their casebase entries.

data is SELECTed from ORACLE tables from a ProC program (EAB_INITIALIZE_IDA_RULEBASE) which will be used as part of two External Action Blocks (file_to_ida_rules and call_record_to_ida_rules).

Internal ART-IM schema are populated by the ProC program

Looking at the creation of Error and Fix data, if errors are detected during incoming data validation which are associated with data that can be fixed then an audit trial of the fixes applied by the rule base needs to be maintained:

for every file structure in error a row entry is created in the FILE_ERROR_LOG table. This is done by the streamer process.

for every call record in error a row entry is created in the CALL_RECORD_ERROR_LOG. This can be done by the streamer process or by ART-IM.

For every error detected and fix applied at the file structure level a row entry is created in the FILE_STRUCTURE_RULE_LOG on the ORACLE database. This is best done by the rule base using a generic rule which is triggered when all file level error detection and fixing has completed. The rule should fire once for each error detected/fix applied and when fired will invoke a user-defined-procedure call sql_exec_limited which does the necessary insertion.

for every error detected and fix applied at the file structure level a row entry is created in the CALL_RECORD_RULE_LOG on the ORACLE database. This is best done by the rule base using a generic rule which is triggered when all call record level error detection and fixing has completed. Again, the rule should fire once for each error detected/fix applied and when fired will invoke a user-defined-procedure call sql-exec-immed which does the necessary insertion.

the ART-IM rules will populate the inserted values from slots on internal schemas.

Looking at the creation of Routing Error Patterns and Closest Matches data, if errors are detected during incoming data validation which are associated with data that is suspended then a record of the incoming call record error pattern (based on TUN, NNI, route group number, route group, direction) together with the three closest matches (based on the closest patterns on the routing reference model to the incoming call record in error) needs to be stored on the ORACLE database for later suspense filed processing. Patterns are stored following completion of all validation/fix processing. In more detail:

for every error generated that is a suspense file error (and assuming no unfixable errors have been generated on the same call record—these unfixable call records are weeded-out using the move-to-sump generic rule), a generic rule (move_to_suspense_file_area) is fired. The rule tries to select the pattern in error from the database, and, if the pattern in error exists:

i) tests for any entry in FILE_ROUTE_ERROR_LINK with relevant pattern exchange file and foreign keys.

ii) if the entry exists no further action is required.

iii) if the entry does not exist then inserts a row entry into the FILE_ROUTE_ERROR_LINK.

If the pattern in error does not exist:

iv) inserts a row entry into a ROUTE_ERROR_PATTERN table populated by error pattern data from incoming call records.

v) inserts a row entry into FILE_ROUTE_ERROR_LINK.

vi) inserts up to 3 row entries into the CLOSEST_MATCHES table populated by routing reference patterns found from previous casebase processing to be closest to the route pattern in error.

A user defined procedure is used to pass SQL command to ORACLE

The ART-IM rules will populate the inserted values from slots on internal schemas.

FIGS. 20, 21, 37 TO 43: USE OF EXPERT SYSTEM BY DATA ANALYZER 7

In the flow diagrams referenced below, it might be noted that a slightly different format has been applied from that of earlier flow diagrams in this specification. That is, function calls are denoted by boxes with double verticle lines, simple statements are denoted by boxes with single vertical lines, and yes/no decisions are denoted by a simple diamond.

The use of the ART-IM expert system by the data analyzer **7** can be expressed In flow diagrams. Referring to FIGS. **16**, **17** and **20**, once it has been determined that there is a failure at call record level, step **1605**, and the next call record error log has been selected from a file, step **1702**, the relevant call records are sent to the expert system, step **2000**. The expert system locates the correct APDU, steps **2005**, **2010**, and then the errored call record, steps **2015**, **2020**.

The expert system then checks whether the call record is correctly itemized (step **2025**), in this example according to System X itemization, and, if it is not, directs the call record to sump by setting the IEF status to "SUMP", step **2030**, while updating the call record error log, step **2035**. If the call record is correctly itemized, it is "put through" the expert system, steps **2040**, **2045**, **2050**, and the results assessed by the data analyzer **7**, in step **1704** onwards.

Referring to FIG. **16** and **21**, it may have been decided that there is failure at file or APDU level, step **1604**. In that case, the file is loaded to memory and the file header and APDUs sent to the expert system; step **2100**. The expert system database is called up, step **2105**, and the APDU schemas from the previous run deleted, step **2110**. The first test run is to refresh the expert system version of the Routing Reference Model, step **2115**, which may immediately result in correcting the apparent error. If not, the default data for the expert system is refreshed, step **2120**, in case for instance default data for the error concerned has previously been missing. If either of these is successful, the data analyzer process reasserts itself, FIG. **16**, and the results from the expert system refresh steps will allow the file to go to validation of its call records, step **1611**. If neither is successful, the call records themselves must be in individually validated. This is described below.

Referring to FIG. **37**, the function box **2125** of FIG. **21** "map header and APDU schema", expands to include loading (steps **3700** to **3725**, **3735**) and running (steps **3730**, **3740**, **3745**, **3750**) the expert system, ART-IM, with respect to call records from the errored files which could not be processed successfully after refreshes of the Routing Reference Model and default data. This loading process includes getting data not available on the ART database ("Foreign Keys"), for instance data from the Streamer **6**, in step **3715**, to enable the expert system so access the files. Having analyzed each call record, the ART supplies as status (step **3755**), which may indicate the call record is fixed or should be suspended or sumped. The data analyzer process (IEF) keeps a count of the call records to be sumped, step **3760**, and sets a flag in the ART-IM, step **3765**, which triggers clean-up by the ART-IM, step **3770**, to clear out each call record and relevant schemas to avoid these simply building up.

Referring to FIGS. **38** to **43**, the application of the expert system file rules can also be expressed in flow diagrams, and the following examples are shown, the flow diagrams being self-explanatory:

i) FIG. **38**; ART File Rules (exchange file header)
- This can be applied to
- trailer APDU
- format version number
- filter type
- node timestamp
- DDC/NMP timestamp (NMP stands for Network Mediation Processor)
- class of data transfer
- node cluster identity
- streamer NNI
- application group
- part file indicator
- file byte size
- table size
- selected APDU type

ii) FIG. **39**; APDU first sequence number rule

iii) FIG. **40**; APDU last sequence number rule

iv) FIG. **41**; APDU sequence number count rule

v) FIG. **42**; ART APDU rules
- This can be applied to
- retransmission indicator
- linking field

vi) FIG. **43**; ART call record rules
- This can be applied to
- record use
- billed all indicator
- clearing cause
- PBX suffix
- CLI cluster identity
- network circuit
- network band
- circuit identity
- circuit number charge band
- call sampling method
- sampling mode
- count reset indicator
- value of N (where N relates to a count made while running a test set of call records for example)
- called party clear timestamp

8. FIGS. 36 AND 44; COMPANY SYSTEM

Referring to FIG. **4**, the output from the Streamer **6** to the company System **8** comprises call records sorted according to billable entity, and validated as described above using a data analyzer incorporation the ART-IM expert system.

The primary role of the Company System **8** is to price the call records and to output the priced records so that they can be billed to clients. However, it also has a validation role, as mentioned above, with emphasis on data relevant to the billable entity and the relationship between the billable entity and the operator of the first network **1**. The company system **8** therefore incorporates or accesses a company system data analyzer, referred to in the following as "cIDA".

The cIDA application can reside alongside the data analyzer **7** which validates data from the Streamer **6**, described above. In FIG. **4**, the steps of fixing errored call records, **430**, bulking the fixed call records, **440**, and investigating unfixable call records, **450**, can all be carried out by means of the cIDA application.

Interestingly, it has been noted that the majority of errors of the order of 90% of the errors picked up by the company system **8**, concern decode anomalies, mainly to do with "time lines" such as "123" and "emergency services" (999) calls. The bulk of the remainder of errors can be attributed to discrepancies in reference data. There can therefore be two primary aspects to building a data analyzer for use with the company system **8**, these being to tacked records providing the majority of the errors, the decode anomalies, and then to provide an infrastructure capable of representing files back to the company system **8** after correction.

Processing Overview

A suitable arrangement might be as follows. Error and warning files are sent from the company box **8** to the cIDA where they are loaded to specific directories, one per operator. A single file can hold zero or many records. Preferably, the cIDA provides a parallel processing facility for all operators, running concurrently, with the capability of manual override. A log is maintained in order to control the sequence of files into and out of the cIDA.

Once an error file has been selected for processing, the cIDA selects each record in turn, assuming the file is not empty, and evaluates the error into one of two categories: fixable and unfixable. Unfixable records are written to a table, reported on, and can later be removed from the database for archiving. Where a record has been deemed to be fixable, it might be fixed automatically by applying rules, or it might need manual intervention before it can be fixed.

Each record, irrespective of error type, is inserted into an ORACLE database table, with all details passed from the company box **8** and a flag set to indicate the "state". The state might, in accordance with the above, be selected from
- suspense
- unfixable
- rules

Users, using Business Objects run at regular intervals, have the capability to view all records currently held and the state allocation they have been given. An audit log can be held for a relevant period, such as for one month for all "charging number string" corrections.

It might be noted that the use of automatic rules may well be found unnecessary. By correcting errors caused by decode anomalies, that is 90% of current errors, the error rate has been found to be reduced to 0.01%. Hence, the simplicity of errors arising means that a system employing automatic rules would be over complicated.

Referring to FIG. **44**, the dataflow routes about the data collection and processing system of the present invention

can be seen. In this Figure, data stores such as files and tables are represented by the horizontally extending rectangles with vertical dotted lines, and processes are represented by the bigger blocks, incorporating rectangles. Entities external to the whole system, such as the NCDB **9**, are represented by the "lozenges".

As already described, raw call data is entered to the Streamer, which converts the raw call data, validates and processes the call records, involving a data analyzer so far as necessary, and outputs validated, itemized call records to the company box. The company box firstly performs operator specific validation, and secondly aggregates itemized call records. At this stage, the call records are priced, using charging information for instance from the national charging database (NCDB) **9**, and output in summarized form to produce a bill report for the relevant client system **10**. Other outputs include the expanded call records, stored in optical disc **71**, and summarized call records for a management reporting system **4400**.

It can be seen in FIG. **44** that there is also an output from the data analyzer to an auditing system "CARDVU" **4405**.

Although embodiments of the present invention can provide extremely detailed information for audit purposes, the auditing system itself is not part of the invention and is not therefore described herein, beyond the comments below at "9. AUDIT TRAIL".

Referring to FIG. **36**, a data model for the company system **8** shows clearly the data sources for use at charging and pricing by the company system **8**. Much the greatest amount of data, the "C&P reference data", is derived from the NCDB **9**. However, there are constraints set by the accounting agreement **4500** between the billable entity and the operator of network **1**. Many issues can be dealt with from the network management center and the data model of FIG. **36** provides appropriate visibility thereto by means of the "telecoms network operator role", box **4505**.

The following initials, used in FIG. **36**, can be expanded as follows:

CBM	Charge Band Matrix
CB	Charge Band
NN	Kingston Communications, Hull (an operator in the UK of a network interconnected to the BT PSTN
TE	Telecom Eirann (as above)
NCIP	National Charging Information Package (an interface to data on the NCDB)

Pricing and charging engines, complying with the type of constraints offered by the system of the present invention, are known and specific description of the charging and pricing engine is not therefore offered here. Indeed, although the data model of FIG. **36** shows all entities involved, not all the relationships are shown as the representation would become to complicated. Overall, however, it must be borne in mind that the call records handled by the company system **8** are already sorted according to billable entity. The aspect of the data needs to be maintained, clearly, so that relevant reports can be allocated to the correct client systems **10**. This can be done, as indicated above, for instance by maintaining allocated directories for the billable entities.

9. Audit Trail

An arrangement as described above can provide a sophisticated audit trail. Data from the exchange at the point of interconnect comes in a file, and is packaged into APDUs. The streamer system **6** polls data off the DDCs **5** using the FTAM protocol, The data being in binary, in call records. The streamer system **6** validates the data against the data base containing reference data, the Routing Reference Model, and assesses which other network operator should be billed. The streamer system **6** writes a full call record in ASCII with operator and exchange information added.

An audit trail arises as follows. On the exchange, call instances are numbered with a File Generation Number which cycles from 0–9999. The DDC **5** also adds a sequence number which cycles from 0–999999, at the file level. Within the file, APDUs are also sequenced with an APDU sequence number which cycles from 0–16353, being binary.

This means that there is stored a record of the number of records in a file, the APDU start and finish numbers, and the number of APDUs.

Because a sequence number is added to each number at the exchange, it can be ensured that the company box **8** receives the numbers in sequence, although they will not necessarily be processed in order. The streamer system **6** actually processes in parallel from different exchanges at the same time.

In the data analyzer, where a "pattern net" is used, by means of which data will "fire" a rule if it does not fit valid content, the analyzer can patch data items only where the data item concerned would not affect price or the audit trail. Patch in this context means set to a standard value. Hence, the data analyzer cannot change the call record sequence number because that identifies the call record. If the call record sequence number were to be changed, there would be no audit trail.

The system described above is, as stated, only one specific embodiment of the invention. It relates to a PSTN and, as described, deals with call records in a voice communications system. Further, the specific form of call records involved, System X Type **6**, relate to only one type of exchange which might be used at a point of interconnection (POI) between networks.

Many changes might be made, however, without departing from the spirit of the present invention. A simple extension of the application of the invention is that, as well as using call record data to generate billing information, traffic analysis information can also be picked up and processed. For instance, calls which are ineffective in reaching a destination, "ineffectives", can be counted by the exchange at the POI and the "bulked" outcome input to the data processing system.

However, more significant changes might include the use of the system with communications other than voice communications, even excluding voice communications, and, as already mentioned, it is clearly not essential that a PSTN is involved, although the benefit of embodiments of the invention is clearly significant with a PSTN in the light of the sheer volume of records and complexity of sources involved.

I claim:

1. A process for collecting and processing data in a first communication network, the data concerning communication instances, wherein the network includes at least one respective point of connection to at least one other communications network, the process comprising the steps of:

i) collecting data at a data access point at each said point of connection, said data concerning a communication instance arising in an originating network other than said first network, and comprising route information identifying the originating network and at least one parameter measurement with respect to said communication instance;

ii) transmitting said data into a data processing system;

iii) processing said data to generate billing information;

iv) allocating said billing information to one of said communications networks; and

v) accumulating respective billing information for each of said communications networks[.];

wherein said data processing system includes a data analyzer, and said processing step includes validating the data followed by analyzing invalid data, correcting said invalid data and processing said corrected invalid data as valid data.

2. A process as in claim **1**, wherein said first network comprises a public switched telephone network.

3. A process as in claim **1** wherein said processing step comprises streaming said data according to the identity of said originating network.

4. A process as in claim **1** wherein the first network comprises a communication network including both local exchanges and trunk exchanges and the data processing system includes correlating pricing and charging data from a database in accordance with the route information identifying the originating network.

5. A process in claim **4** wherein said correlation is carried out subsequent to streaming the data.

6. A process as in claim **1** wherein [said data processing system comprises a data analyzer, and said processing step includes validating the data followed by analyzing invalid data,] the analysis [including] *includes* a step of identifying data which can potentially be set to a default value, *and correcting said invalid data includes* setting the data to a default value [and processing it as valid data.]

7. A data processing arrangement for processing data collected in a communications network but concerning call instances arising outside the network, the arrangement comprising:

i) a data input for inputting said data, said data comprising at least one of a plurality of sort characteristics;

ii) verifying means for check the data received at the data input;

iii) a data analyzer for analyzing data rejected by the verifying means, and for substituting amended or default data therefor;

iv) pricing means for pricing data output by the verifying means or by the data analyzer in accordance with updatable reference information;

v) output means for outputting priced data from the pricing means into memory locations, each memory location being dedicated to data relevant to one or more of said sort characteristics, and

(vi) accumulation means for accumulating price data in respect of each communication network causing said call instances.

8. A data processing arrangement as in claim **7** wherein each sort characteristic identifies a further network outside said communications network in which further network an associated communication arose.

9. An arrangement as in claim **7** wherein said communications network is a PSTN.

10. An arrangement as in claim **7** wherein said data analyzer comprises means for storing data which cannot be amended or defaulted in a suspended data store, for potential subsequent processing.

11. A data analyzer for use in a data processing arrangement according to claim **7**.

12. A data collection and processing arrangement for use in a first communication network which is connected to and receives communication instances from multiple further networks, the arrangement comprising:

a) registering means for registering a communication instance incoming to the first network having arisen in one of said further networks,

b) means for formatting a record of said communication instance, the record comprising data identifying said one of the further networks and a parameter value associated with the communication instance,

c) validating means for validating said record,

d) pricing, and charging means for associating pricing and charging data with a validated record and providing a sorted array of priced, charged and validated records, the array being, sorted according, to the identities of the further networks, and

e) analyzing means for analyzing records which are rejected by the validating means, the analyzing means dealing with the rejected records in one of at least three ways according to the cause of rejection, said three ways being:

i) to set values in a non-validated record (NVR) to a best-fit value,

ii) to set values in a NVR to default values; and

iii) to archive or dump the NVR;

records which have been dealt with in either of the ways i) or ii) being transmitted, directly or indirectly, to the pricing and charging means as validated records.

13. An arrangement as in claim **12** wherein:

a communication instance is received by an exchange of said first network,

a record of the communication instance being transmitted to a data collector as said registering means,

the data identifying said one of the further networks being provided by routing information incorporated in said record, and

wherein the validating means has access to a routing reference data model and one of the criteria used in reference data model and one of the criteria used in validating a record is the degree of correlation between the routing information and the routing reference data model.

14. An arrangement as in claim **12**, wherein the analyzing means deals with rejected records in one of at least four ways, the four ways comprising i) to iii) and, iv), to append data concerning a NVR to a file in a suspended data store which can be accessed and analyzed at a later time.

15. An arrangement as in claim **14** wherein each file in the suspended data store is dedicated to NVRs having the same error pattern.

16. An arrangement as in claim **12** wherein the pricing and charging means comprises validating means, or access to validating means, and can output non-validated records to the analyzing means so as to allow representing of data which has become corrupted since first being validated in the arrangement.

17. A data collection and processing system for use in collecting and processing communication records relevant to a plurality of networks, wherein said system comprises:

at least one input for communication records generated at a point of connection between a first of said plurality of networks and at least one other of said plurality of networks,

said records providing identification of the network in which an associated communication instance arose or from which it entered said first network,

validation means for validating format and routing information aspects of the records,

data analyzing means for analyzing errored records rejected by said validation means,

the analyzing means being capable of categorizing said errored records and applying default values to at least one category of the errored records,

data sorting means for sorting validated and [defaulted records] *errored records having default values applied thereto* according to said network identification, *and*

pricing means for receiving the sorted records and generating billing information for use in billing entities relevant to the identified networks; and accumulation means for accumulating price data in respect of each network.

18. A process for collecting and processing data in a first communications network, the first communications network comprising a plurality of switches in common ownership of a first party, the data concerning communications instances, wherein the network includes at least one point of connection to a second communications network, the second communications network comprising a further plurality of switches in common ownership of another party, the process comprising the steps of:

i) collecting data at a data access point at said point of connection, said data concerning a communication instance arising in an originating network other than said first network, and comprising route information identifying the originating network and at least one parameter measurement with respect to said communication instance;

ii) transmitting said data into a data processing system; and

iii) processing said data[.];

wherein said data processing system includes a data analyzer, and said processing step includes validating the data followed by analzying invalid data, correcting said invalid data and processing said corrected invalid data as valid data.

19. A process as in claim **18** wherein said first network comprises a public switched telephone network.

20. A process as in claim **18** wherein said processing step comprises streaming said data according to the identity of said originating network.

21. A process as in claim **18** wherein the first network comprises a communications network including both local exchanges and trunk exchanges and the data processing system includes correlating pricing and charging data from a database in accordance with the route information identifying the originating network.

22. A process as in claim **21** wherein said correlation is carried out subsequent to streaming the data.

23. A process as in claim **18** wherein [said data processing system comprises a data analyzer and said processing step includes validating the data followed by analyzing invalid data,] the analysis [including] *includes* a step of identifying data which can potentially be set to a default value, *and correcting said invalid data includes* setting the data to a default value [and processing it as valid data].

24. A data processing arrangement for collecting and processing data in a first communications network, the first communications network comprising a plurality of switches in common ownership of a first party, the data concerning communications instances, wherein the network includes at least one point of connection to a second communications network, the second communications network comprising a further plurality of switches in common ownership of another party, the arrangement comprising:

i) a data input for inputting said data, said data comprising at least one of a plurality of sort characteristics;

ii) verifying means for checking the data received at the data input;

iii) a data analyzer for analyzing data rejected by the verifying means, and for substituting amended or default data therefor;

iv) pricing means for pricing data output by the verifying means or by the data analyzer in accordance with updatable reference information; and

v) output means for outputting priced data from the pricing means into memory locations, each memory location being dedicated to data relevant to one or more of said sort characteristics.

25. A data processing arrangement as in claim **24** wherein each sort characteristics identifies a further network outside said communications network in which further network an associated communication arose.

26. An arrangement as in claim **24** wherein said communications network is a PSTN.

27. An arrangement as in claim **24** wherein said data analyzer comprises means for storing data which cannot be amended or defaulted in a suspended data store, for potential subsequent processing.

28. A data analyzer for use in a data processing arrangement according to claim **24**.

29. A data collection and processing system for collecting and processing data in a first communications network, the first communications network comprising a plurality of switches in common ownership of a first party, the data concerning communications instances, wherein the network includes at least one point of connection to a second communications network, the second communications network comprising a further plurality of switches in common ownership of another party, the system comprising:

at least one input for communication records generated at a point of connection between a first of said plurality of networks and at least one other of said plurality of networks,

said records providing identification of the network in which an associated communication instance arose or from which it entered said first network,

validation means for validating format and routing information aspects of the records,

data analyzing means for analyzing errored records rejected by said validation means,

the analyzing means being capable of categorizing said errored records and applying default values to at least one category of the errored records,

data sorting means for sorting validated and [defaulted records] *errored records having said default values applied thereto* according to said network identification, and

pricing means for receiving the sorted records and generating billing information for use in billing entities relevant to the identified networks.

* * * * *

US005403639A

United States Patent [19]

Belsan et al.

[11] Patent Number: 5,403,639

[45] Date of Patent: Apr. 4, 1995

[54] **FILE SERVER HAVING SNAPSHOT APPLICATION DATA GROUPS**

[75] Inventors: **Jay S. Belsan; Jeffrey S. Laughlin,** both of Nederland; **Mogens H. Pedersen,** Longmont; **Robert J. Raicer,** Niwot; **George A. Rudeseal,** Boulder; **Charles P. Schafer,** Louisville; **Barbara L. Steele,** Boulder; **Patrick J. Tomsula,** Arvada, all of Colo.

[73] Assignee: **Storage Technology Corporation,** Louisville, Colo.

[21] Appl. No.: **939,312**

[22] Filed: **Sep. 2, 1992**

[51] **Int. Cl.**[6] **G06F 15/40**

[52] **U.S. Cl.** **395/600**; 395/425; 395/800; 395/650; 364/DIG. 2

[58] **Field of Search** 395/600, 425, 800, 650, 395/400

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,627,019	12/1986	Ng	395/425
5,124,987	6/1992	Milligan et al.	371/10.1
5,155,835	10/1992	Beisan	395/425
5,193,184	3/1993	Belsan et al.	395/600
5,210,866	5/1993	Milligan et al.	364/DIG. 2
5,212,789	5/1993	Ragu	395/600
5,239,659	8/1993	Rudeseal et al.	395/800
5,247,647	9/1993	Brow et al.	395/600
5,278,979	1/1994	Foster et al.	395/600
5,287,496	2/1994	Chen et al.	395/600

Primary Examiner—Thomas G. Black
Assistant Examiner—Cuan Pham
Attorney, Agent, or Firm—Duft, Graziano & Forest

[57] **ABSTRACT**

This file server system appears to the host computer to be a plurality of data storage devices which are directly addressable by the host computer using the native data management and access structures of the host computer. The file server however is an intelligent data storage subsystem that defines, manages and accesses synchronized sets of data and maintains these synchronized sets of data external from the host computer system's data management facilities in a manner that is completely transparent to the host computer. This is accomplished by the use of the snapshot application data group that extends the traditional sequential data set processing concept of generation data groups.

42 Claims, 19 Drawing Sheets

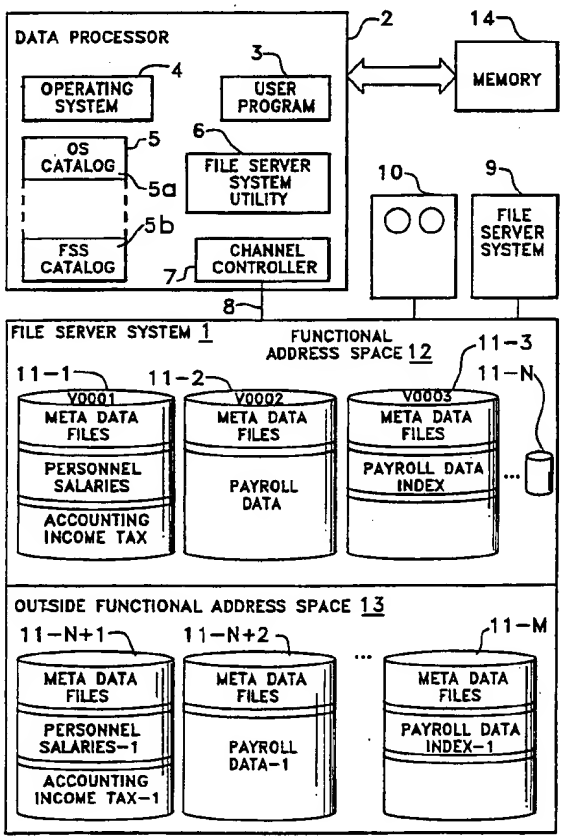
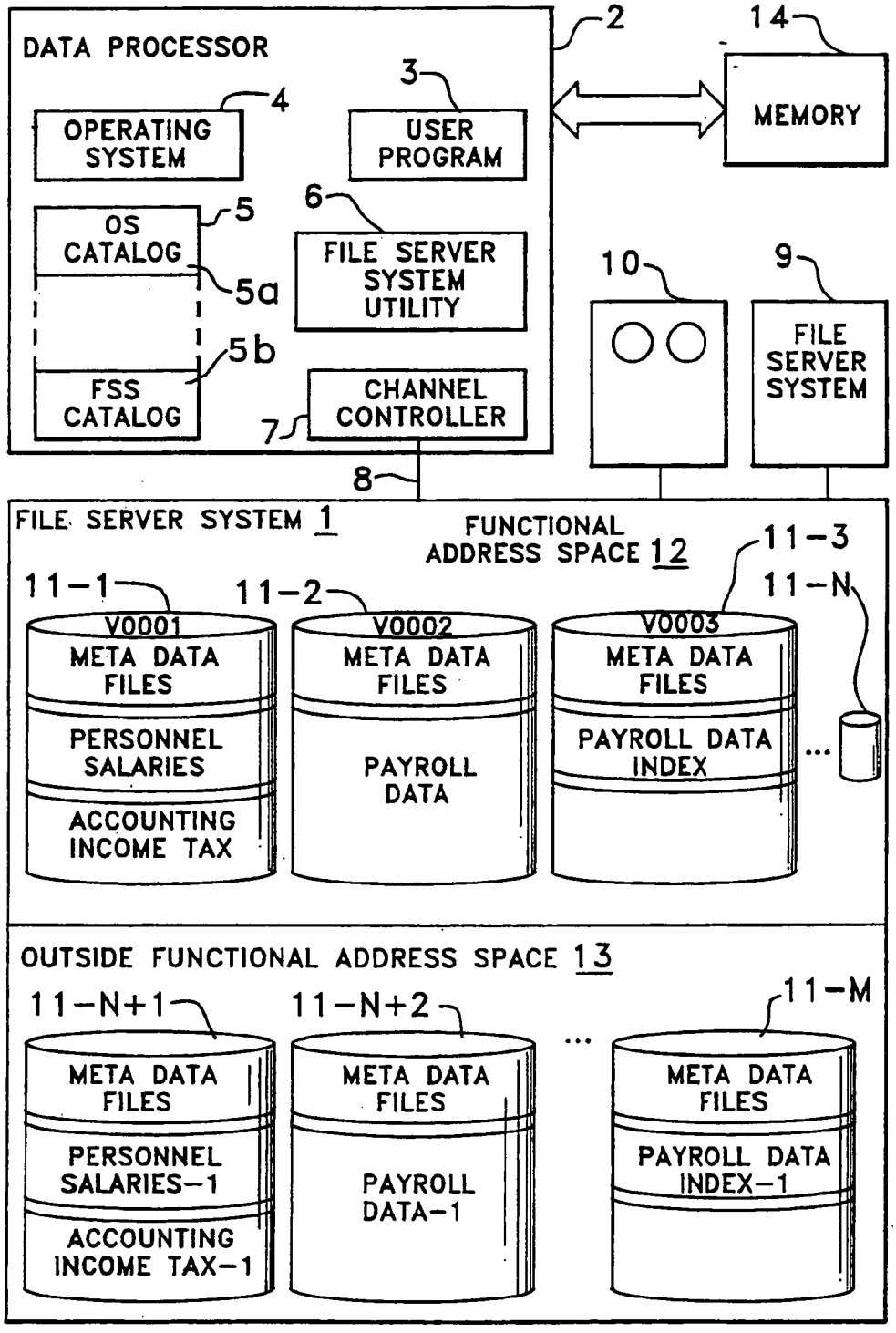


FIG. 1.


DATA PROCESSOR
2
14
3
4
OPERATING SYSTEM
USER PROGRAM
MEMORY
6
5
OS CATALOG
FILE SERVER SYSTEM UTILITY
10
9
5a
5b
FSS CATALOG
CHANNEL CONTROLLER
7
FILE SERVER SYSTEM
8
FILE SERVER SYSTEM 1
FUNCTIONAL ADDRESS SPACE 12
11-1
11-2
11-3
11-N
V0001
META DATA FILES
PERSONNEL SALARIES
ACCOUNTING INCOME TAX
V0002
META DATA FILES
PAYROLL DATA
V0003
META DATA FILES
PAYROLL DATA INDEX
OUTSIDE FUNCTIONAL ADDRESS SPACE 13
11-N+1
11-N+2
11-M
META DATA FILES
PERSONNEL SALARIES-1
ACCOUNTING INCOME TAX-1
META DATA FILES
PAYROLL DATA-1
META DATA FILES
PAYROLL DATA INDEX-1

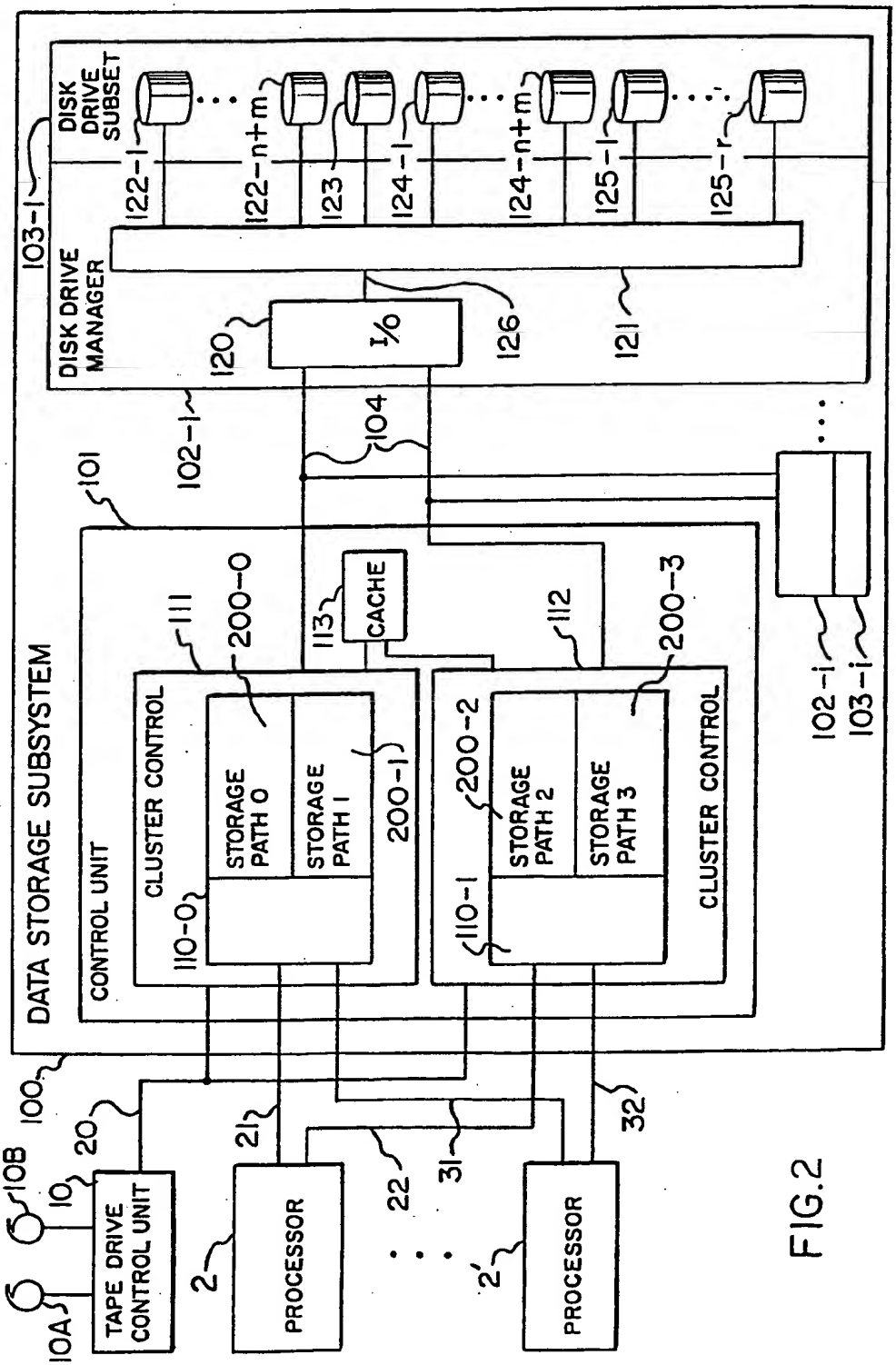

DATA STORAGE SUBSYSTEM
DISK DRIVE SUBSET
DISK DRIVE MANAGER
103-1
122-1
122-n+m
123
124-1
124-n+m
125-1
125-r
120
I/O
126
121
102-1
104
101
CONTROL UNIT
CLUSTER CONTROL
111
200-0
113
CACHE
112
200-3
STORAGE PATH 0
STORAGE PATH 1
200-1
110-0
200-2
STORAGE PATH 2
STORAGE PATH 3
110-1
CLUSTER CONTROL
102-i
103-i
100
20
21
22
31
32
10B
10
10A
TAPE DRIVE CONTROL UNIT
2
PROCESSOR
2'
PROCESSOR

FIG.2

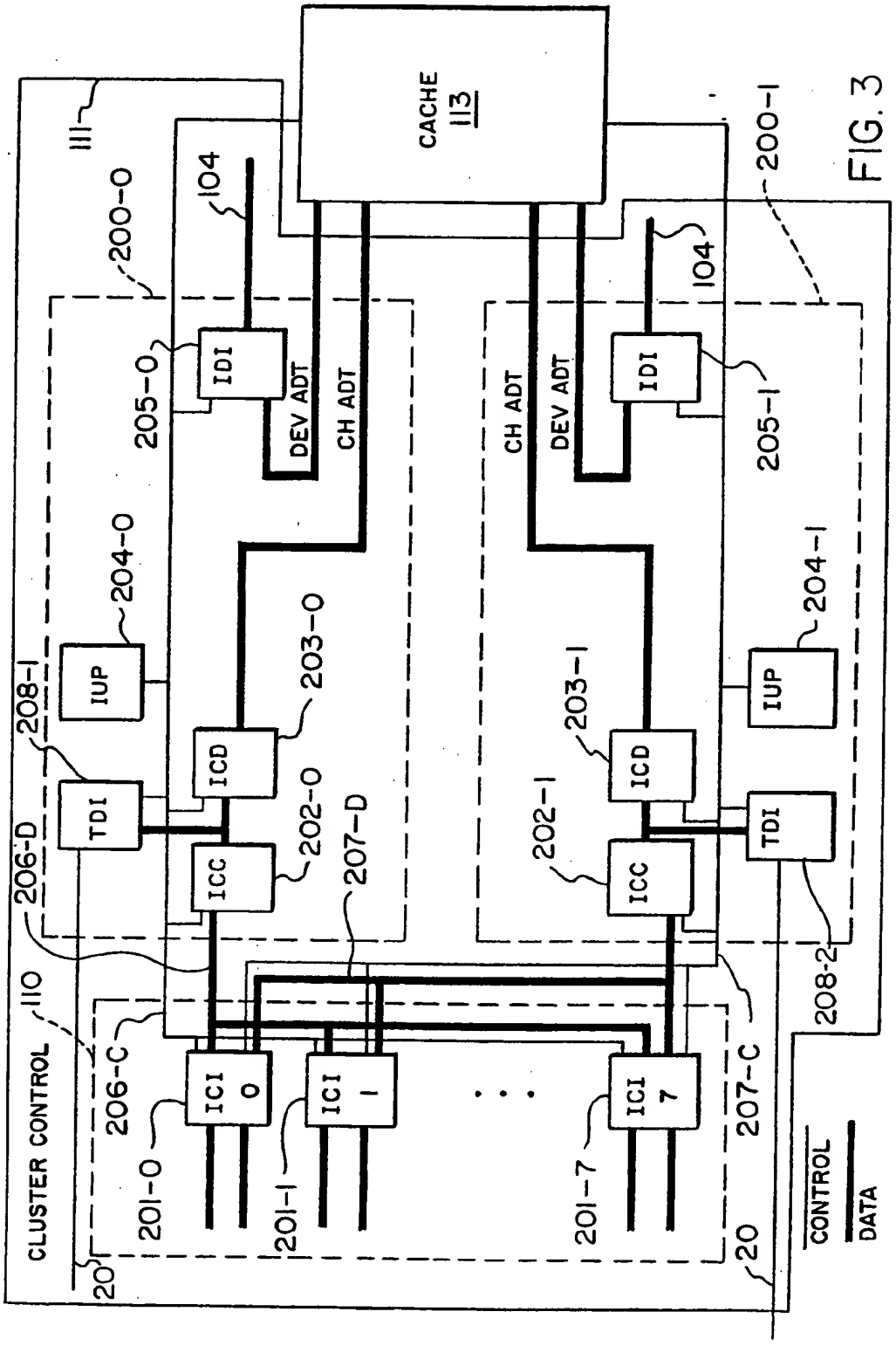

CACHE
113
111
104
200-0
200-1
205-0
205-1
IDI
DEV ADT
CH ADT
204-0
204-1
IUP
203-0
203-1
ICD
208-1
TDI
206-D
202-0
202-1
ICC
207-D
110
CLUSTER CONTROL
206-C
207-C
208-2
ICI 0
ICI 1
ICI 7
201-0
201-1
201-7
20
CONTROL
DATA

FIG. 3

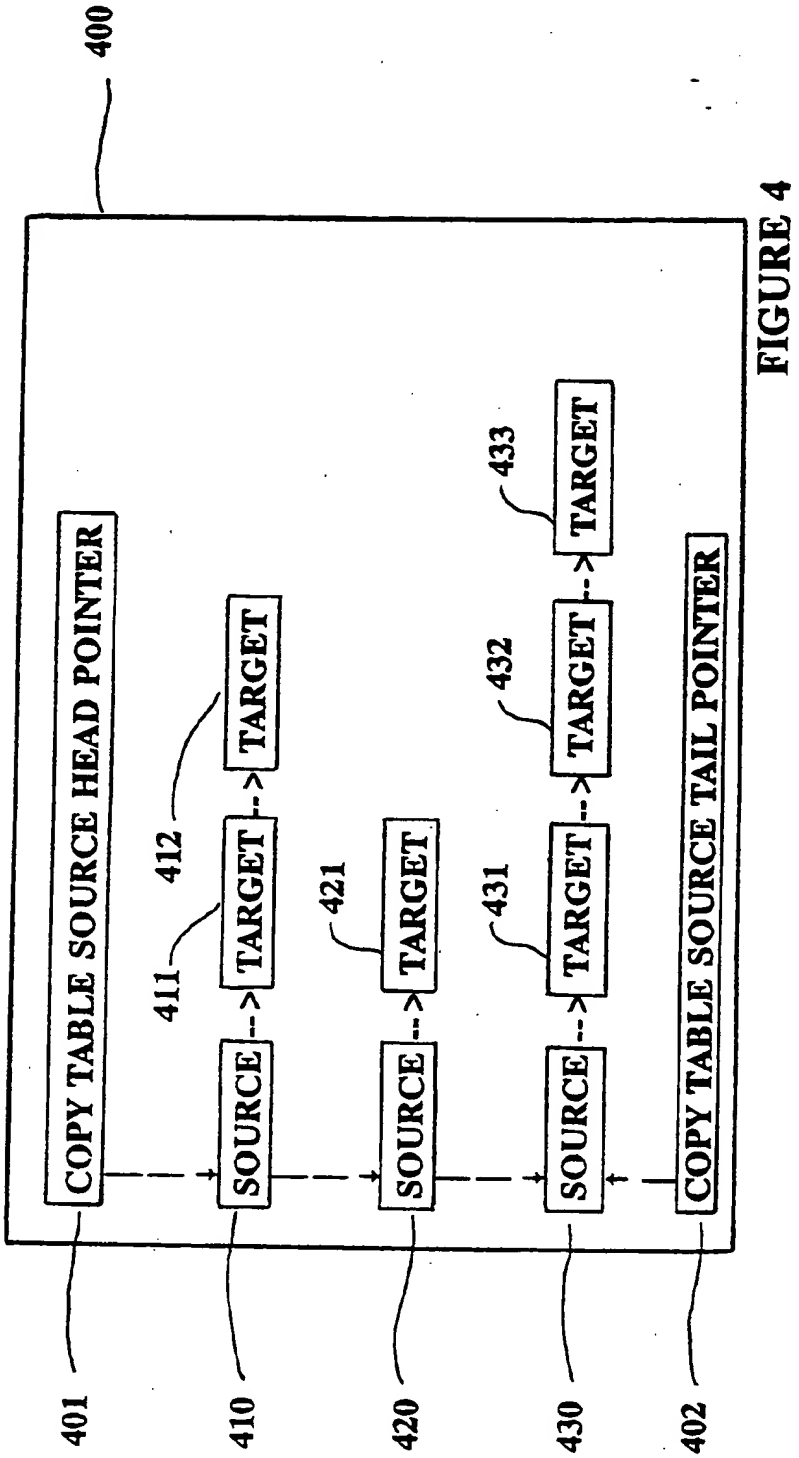

400
COPY TABLE SOURCE HEAD POINTER
SOURCE
TARGET
411
412
421
431
432
433
COPY TABLE SOURCE TAIL POINTER
401
410
420
430
402

FIGURE 4

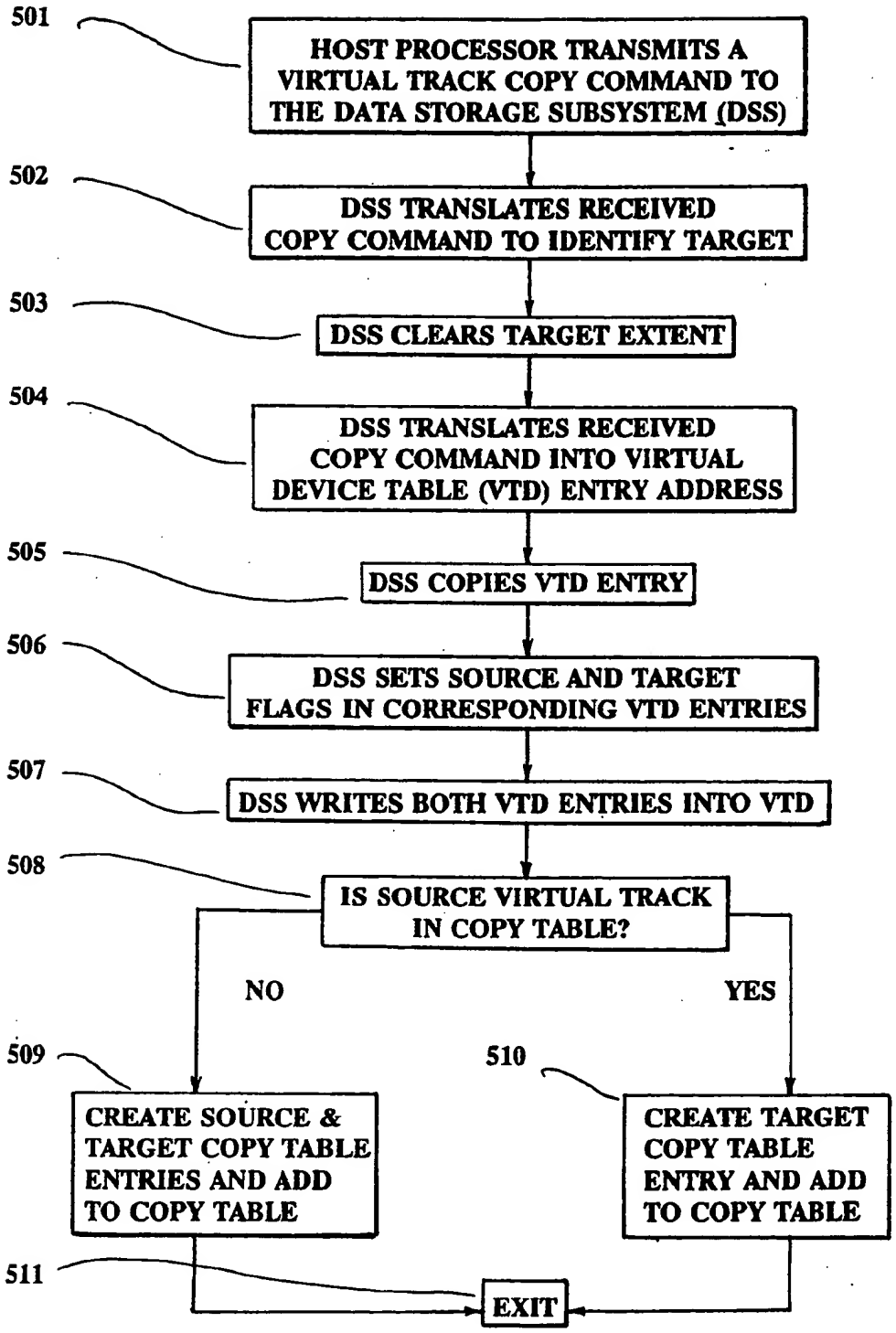

501
HOST PROCESSOR TRANSMITS A VIRTUAL TRACK COPY COMMAND TO THE DATA STORAGE SUBSYSTEM (DSS)
502
DSS TRANSLATES RECEIVED COPY COMMAND TO IDENTIFY TARGET
503
DSS CLEARS TARGET EXTENT
504
DSS TRANSLATES RECEIVED COPY COMMAND INTO VIRTUAL DEVICE TABLE (VTD) ENTRY ADDRESS
505
DSS COPIES VTD ENTRY
506
DSS SETS SOURCE AND TARGET FLAGS IN CORRESPONDING VTD ENTRIES
507
DSS WRITES BOTH VTD ENTRIES INTO VTD
508
IS SOURCE VIRTUAL TRACK IN COPY TABLE?
NO
YES
509
CREATE SOURCE & TARGET COPY TABLE ENTRIES AND ADD TO COPY TABLE
510
CREATE TARGET COPY TABLE ENTRY AND ADD TO COPY TABLE
511
EXIT

FIGURE 5

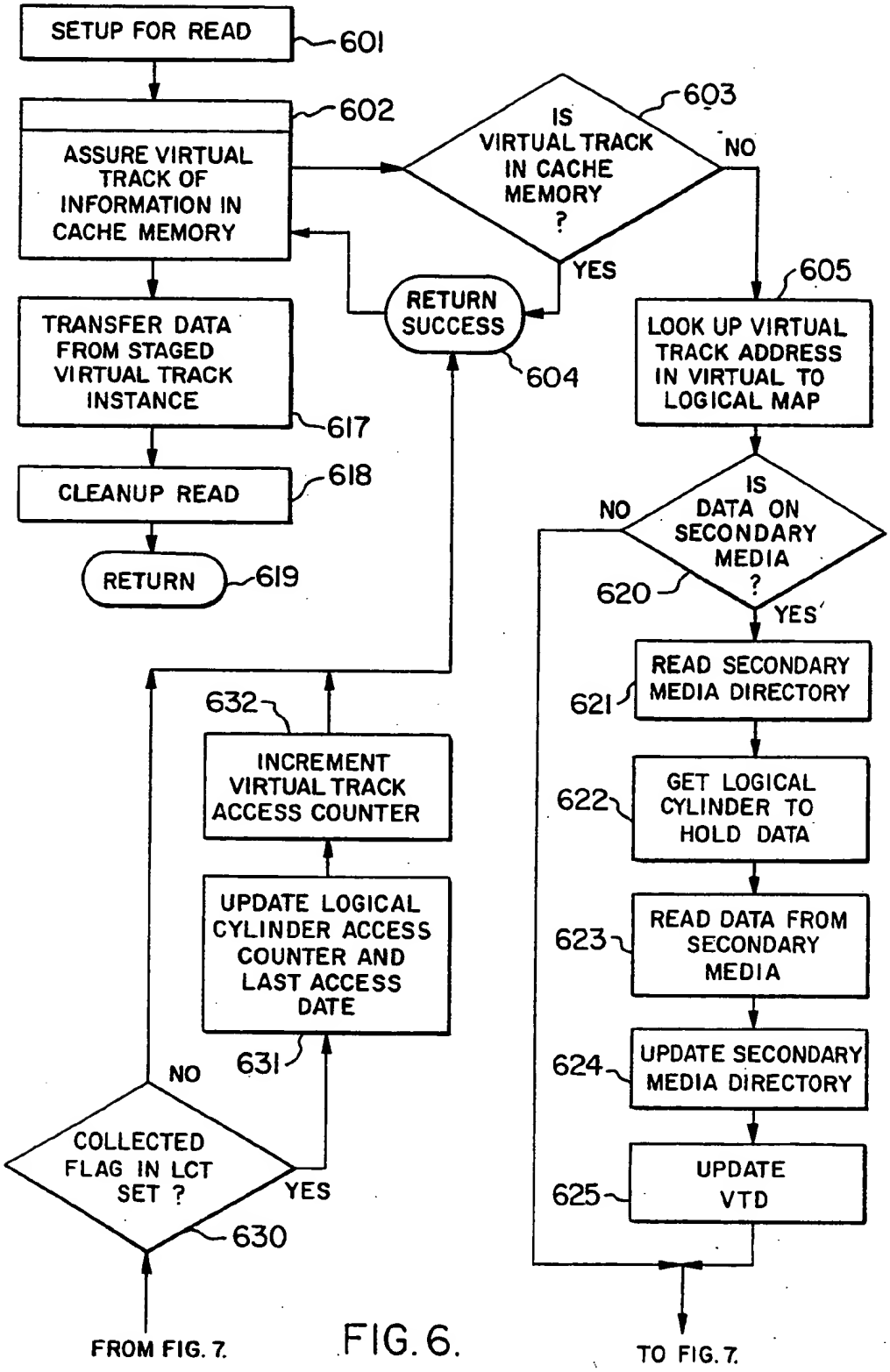

SETUP FOR READ
601
602
ASSURE VIRTUAL TRACK OF INFORMATION IN CACHE MEMORY
603
IS VIRTUAL TRACK IN CACHE MEMORY ?
NO
YES
RETURN SUCCESS
604
605
LOOK UP VIRTUAL TRACK ADDRESS IN VIRTUAL TO LOGICAL MAP
TRANSFER DATA FROM STAGED VIRTUAL TRACK INSTANCE
617
CLEANUP READ
618
RETURN
619
IS DATA ON SECONDARY MEDIA ?
NO
620
YES
READ SECONDARY MEDIA DIRECTORY
621
GET LOGICAL CYLINDER TO HOLD DATA
622
READ DATA FROM SECONDARY MEDIA
623
UPDATE SECONDARY MEDIA DIRECTORY
624
UPDATE VTD
625
632
INCREMENT VIRTUAL TRACK ACCESS COUNTER
UPDATE LOGICAL CYLINDER ACCESS COUNTER AND LAST ACCESS DATE
631
NO
COLLECTED FLAG IN LCT SET ?
YES
630
FROM FIG. 7.
TO FIG. 7.

FIG. 6.

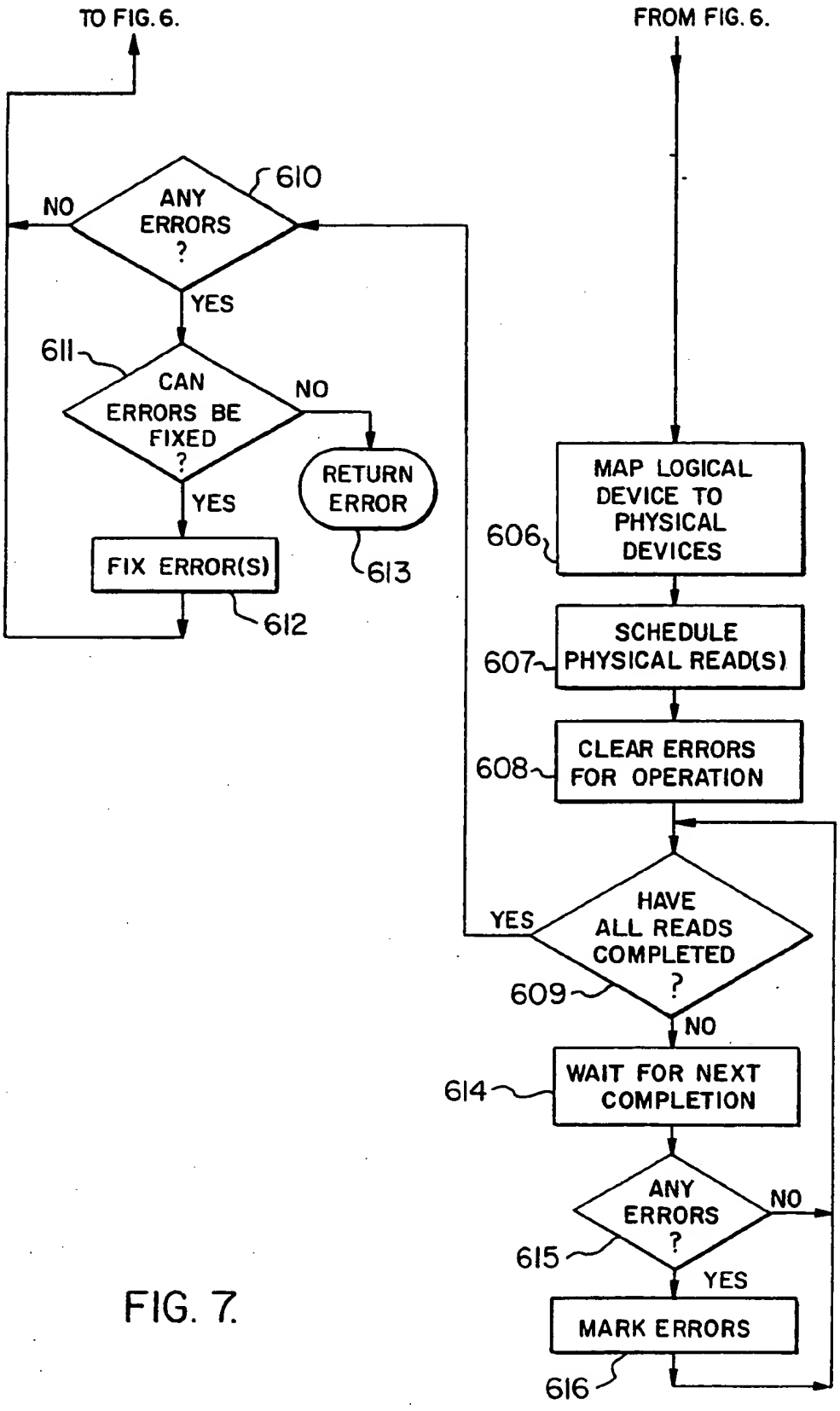

TO FIG. 6.
FROM FIG. 6.
610
ANY ERRORS ?
NO
YES
611
CAN ERRORS BE FIXED ?
NO
YES
RETURN ERROR
613
FIX ERROR(S)
612
MAP LOGICAL DEVICE TO PHYSICAL DEVICES
606
SCHEDULE PHYSICAL READ(S)
607
CLEAR ERRORS FOR OPERATION
608
HAVE ALL READS COMPLETED ?
YES
609
NO
WAIT FOR NEXT COMPLETION
614
ANY ERRORS ?
NO
615
YES
MARK ERRORS
616

FIG. 7.

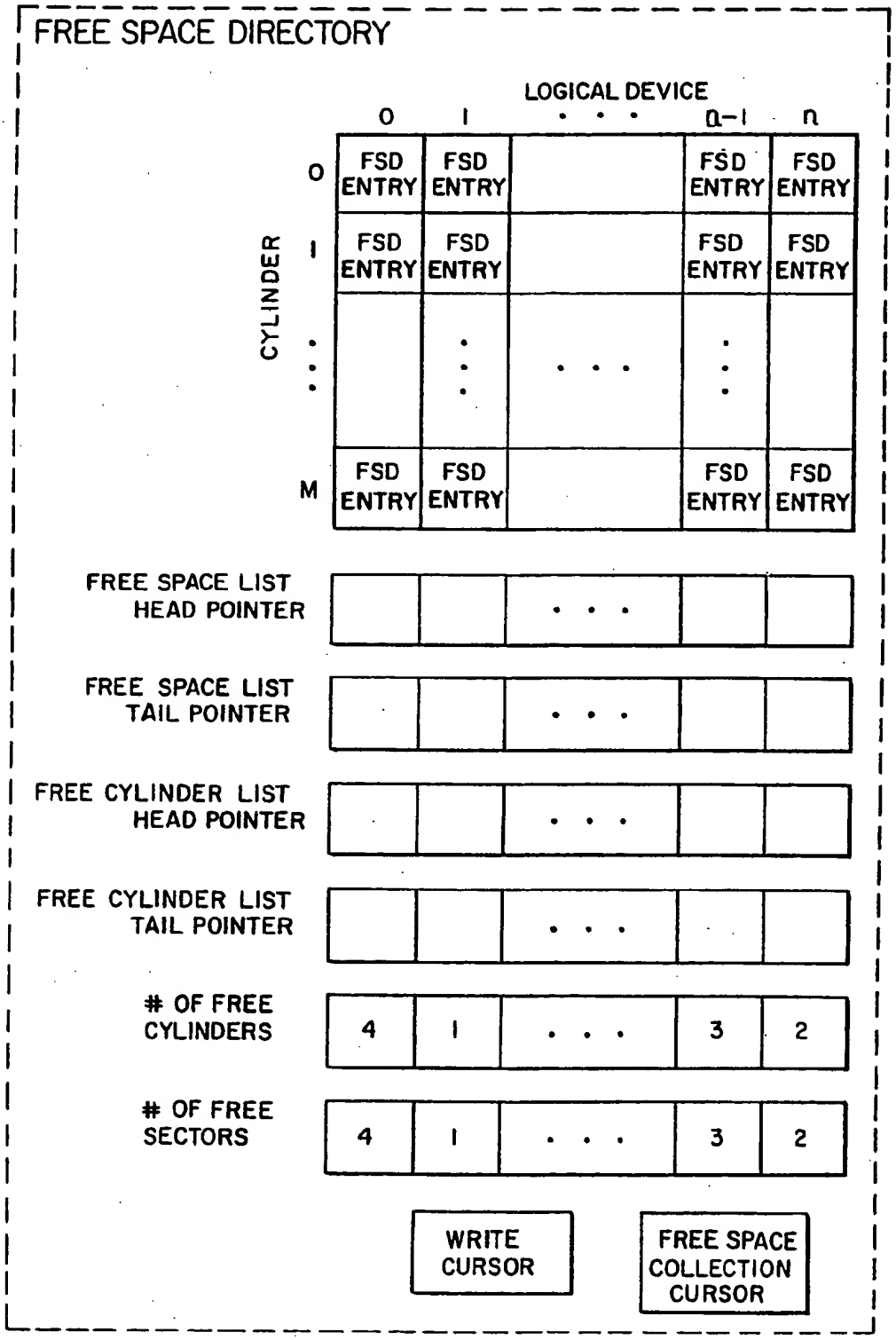

FREE SPACE DIRECTORY
LOGICAL DEVICE
0
1
• • •
n-1
n
CYLINDER
0
1
M
FSD ENTRY
FREE SPACE LIST HEAD POINTER
FREE SPACE LIST TAIL POINTER
FREE CYLINDER LIST HEAD POINTER
FREE CYLINDER LIST TAIL POINTER
OF FREE CYLINDERS
OF FREE SECTORS
4
1
3
2
WRITE CURSOR
FREE SPACE COLLECTION CURSOR

FIG. 8.

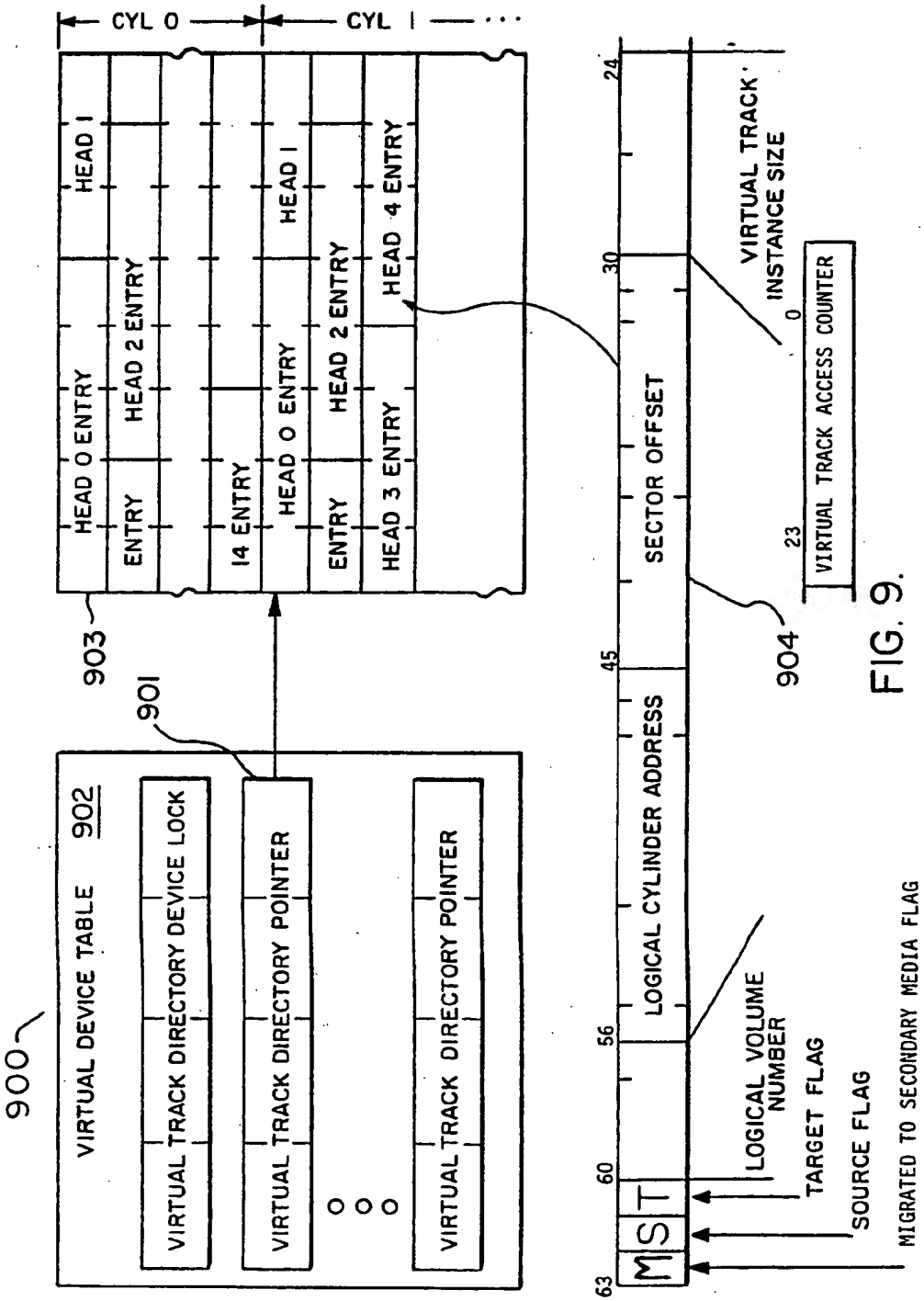

CYL 0
CYL 1
HEAD 0 ENTRY
ENTRY
HEAD 1
HEAD 2 ENTRY
14 ENTRY
HEAD 0 ENTRY
HEAD 1
ENTRY
HEAD 2 ENTRY
HEAD 3 ENTRY
HEAD 4 ENTRY
903
901
900
VIRTUAL DEVICE TABLE 902
VIRTUAL TRACK DIRECTORY DEVICE LOCK
VIRTUAL TRACK DIRECTORY POINTER
VIRTUAL TRACK DIRECTORY POINTER
63
60
56
45
30
24
0
23
MST
LOGICAL CYLINDER ADDRESS
SECTOR OFFSET
VIRTUAL TRACK INSTANCE SIZE
VIRTUAL TRACK ACCESS COUNTER
904
LOGICAL VOLUME NUMBER
TARGET FLAG
SOURCE FLAG
MIGRATED TO SECONDARY MEDIA FLAG

FIG. 9.

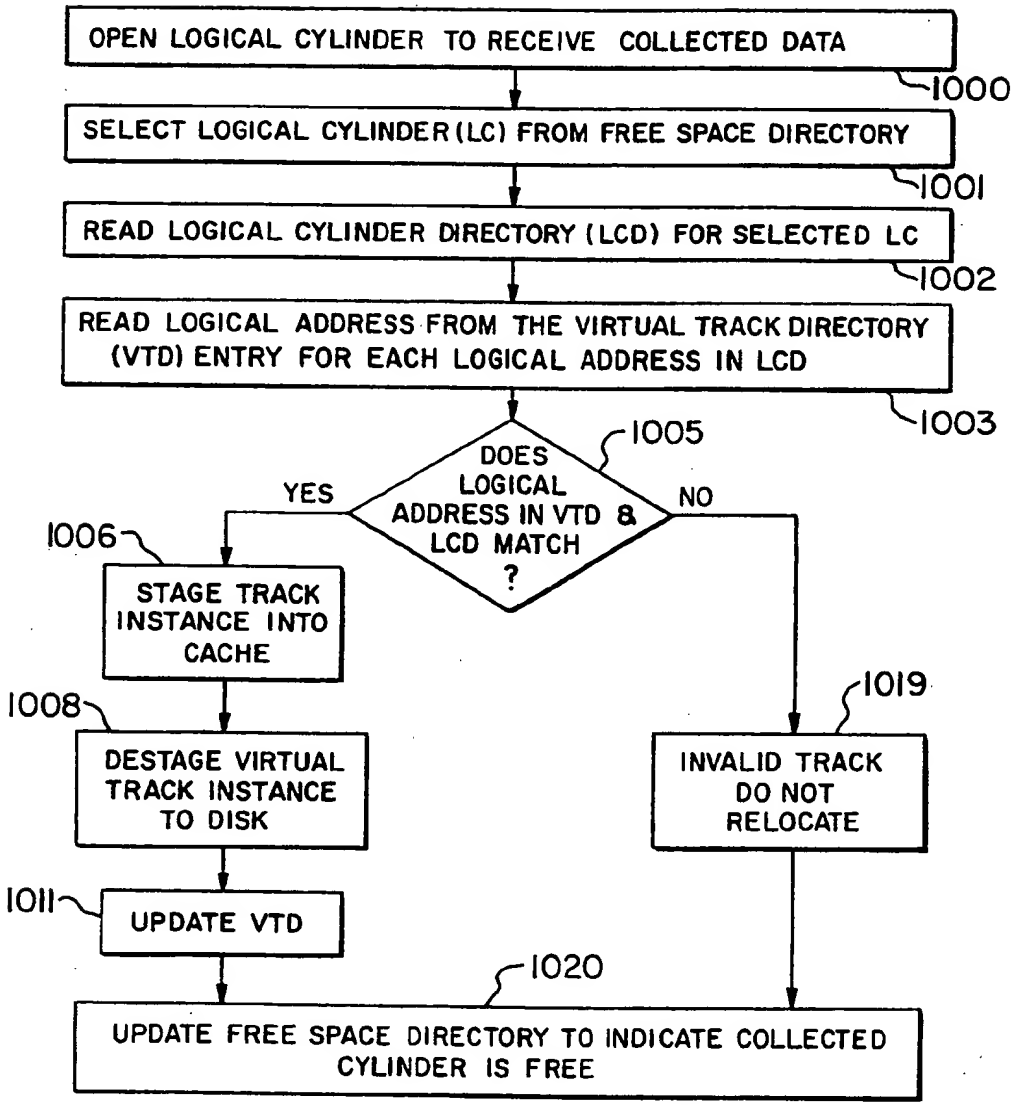

OPEN LOGICAL CYLINDER TO RECEIVE COLLECTED DATA
1000
SELECT LOGICAL CYLINDER (LC) FROM FREE SPACE DIRECTORY
1001
READ LOGICAL CYLINDER DIRECTORY (LCD) FOR SELECTED LC
1002
READ LOGICAL ADDRESS FROM THE VIRTUAL TRACK DIRECTORY (VTD) ENTRY FOR EACH LOGICAL ADDRESS IN LCD
1003
1005
DOES LOGICAL ADDRESS IN VTD & LCD MATCH ?
YES
NO
1006
STAGE TRACK INSTANCE INTO CACHE
1008
DESTAGE VIRTUAL TRACK INSTANCE TO DISK
1011
UPDATE VTD
1019
INVALID TRACK DO NOT RELOCATE
1020
UPDATE FREE SPACE DIRECTORY TO INDICATE COLLECTED CYLINDER IS FREE

FIG. 10.

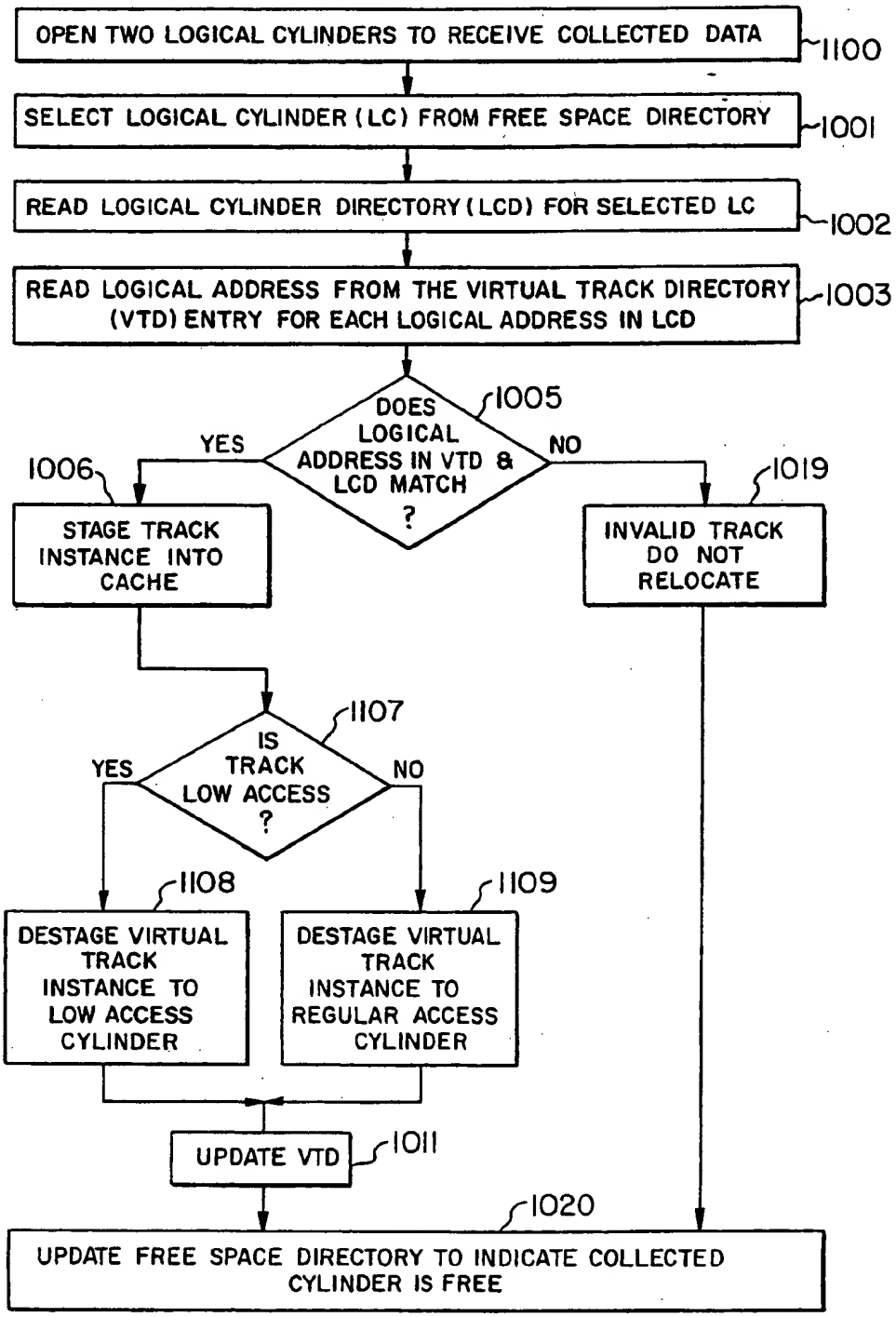

OPEN TWO LOGICAL CYLINDERS TO RECEIVE COLLECTED DATA
1100
SELECT LOGICAL CYLINDER (LC) FROM FREE SPACE DIRECTORY
1001
READ LOGICAL CYLINDER DIRECTORY (LCD) FOR SELECTED LC
1002
READ LOGICAL ADDRESS FROM THE VIRTUAL TRACK DIRECTORY (VTD) ENTRY FOR EACH LOGICAL ADDRESS IN LCD
1003
DOES LOGICAL ADDRESS IN VTD & LCD MATCH ?
1005
YES
NO
1006
STAGE TRACK INSTANCE INTO CACHE
1019
INVALID TRACK DO NOT RELOCATE
IS TRACK LOW ACCESS ?
1107
YES
NO
1108
DESTAGE VIRTUAL TRACK INSTANCE TO LOW ACCESS CYLINDER
1109
DESTAGE VIRTUAL TRACK INSTANCE TO REGULAR ACCESS CYLINDER
UPDATE VTD
1011
1020
UPDATE FREE SPACE DIRECTORY TO INDICATE COLLECTED CYLINDER IS FREE

FIG. 11.

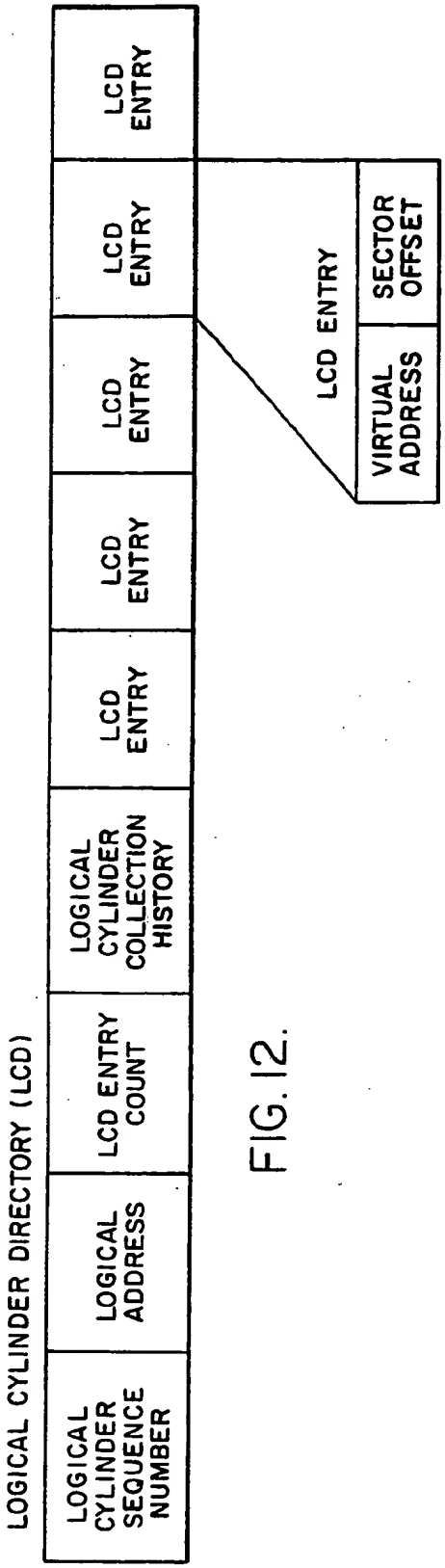

LOGICAL CYLINDER DIRECTORY (LCD)
LOGICAL CYLINDER SEQUENCE NUMBER
LOGICAL ADDRESS
LCD ENTRY COUNT
LOGICAL CYLINDER COLLECTION HISTORY
LCD ENTRY
LCD ENTRY
LCD ENTRY
LCD ENTRY
LCD ENTRY
LCD ENTRY
VIRTUAL ADDRESS
SECTOR OFFSET

FIG. 12.

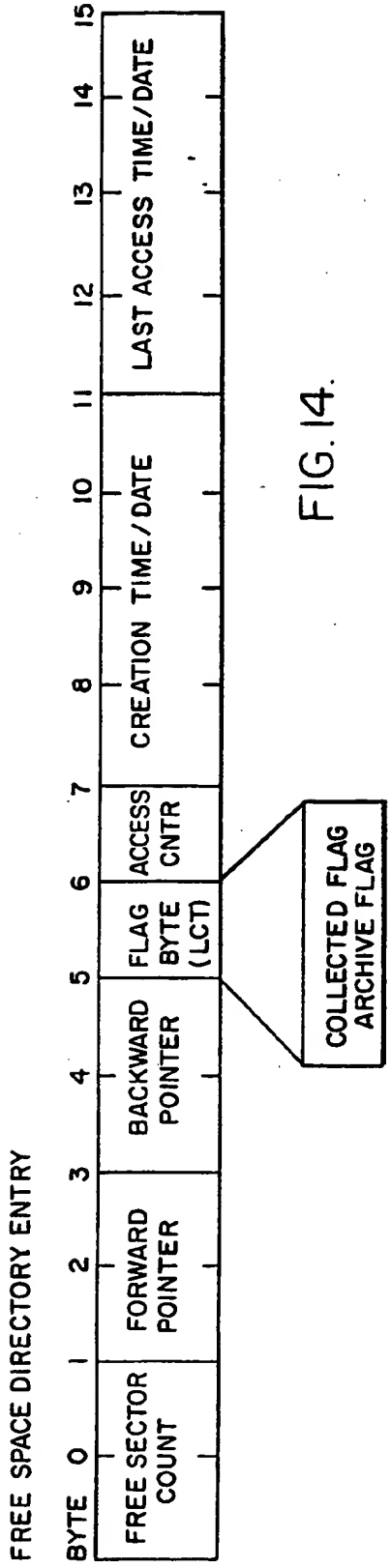

FREE SPACE DIRECTORY ENTRY
BYTE 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
FREE SECTOR COUNT
FORWARD POINTER
BACKWARD POINTER
FLAG BYTE (LCT)
ACCESS CNTR
CREATION TIME/DATE
LAST ACCESS TIME/DATE
COLLECTED FLAG
ARCHIVE FLAG

FIG. 14.

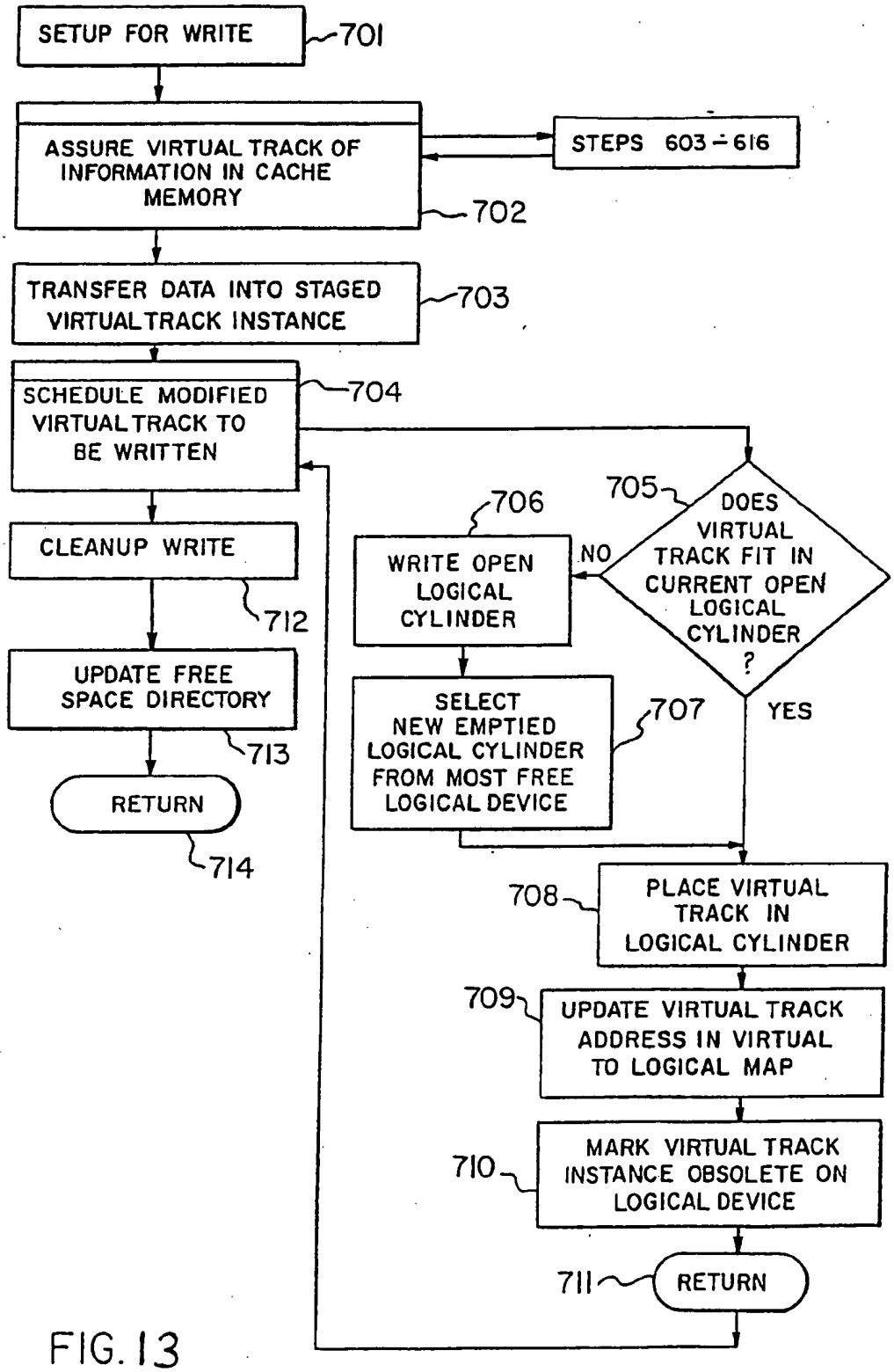

SETUP FOR WRITE
701
ASSURE VIRTUAL TRACK OF INFORMATION IN CACHE MEMORY
STEPS 603 - 616
702
TRANSFER DATA INTO STAGED VIRTUAL TRACK INSTANCE
703
SCHEDULE MODIFIED VIRTUAL TRACK TO BE WRITTEN
704
705
DOES VIRTUAL TRACK FIT IN CURRENT OPEN LOGICAL CYLINDER ?
NO
YES
706
WRITE OPEN LOGICAL CYLINDER
707
SELECT NEW EMPTIED LOGICAL CYLINDER FROM MOST FREE LOGICAL DEVICE
CLEANUP WRITE
712
UPDATE FREE SPACE DIRECTORY
713
RETURN
714
708
PLACE VIRTUAL TRACK IN LOGICAL CYLINDER
709
UPDATE VIRTUAL TRACK ADDRESS IN VIRTUAL TO LOGICAL MAP
710
MARK VIRTUAL TRACK INSTANCE OBSOLETE ON LOGICAL DEVICE
711
RETURN

FIG. 13

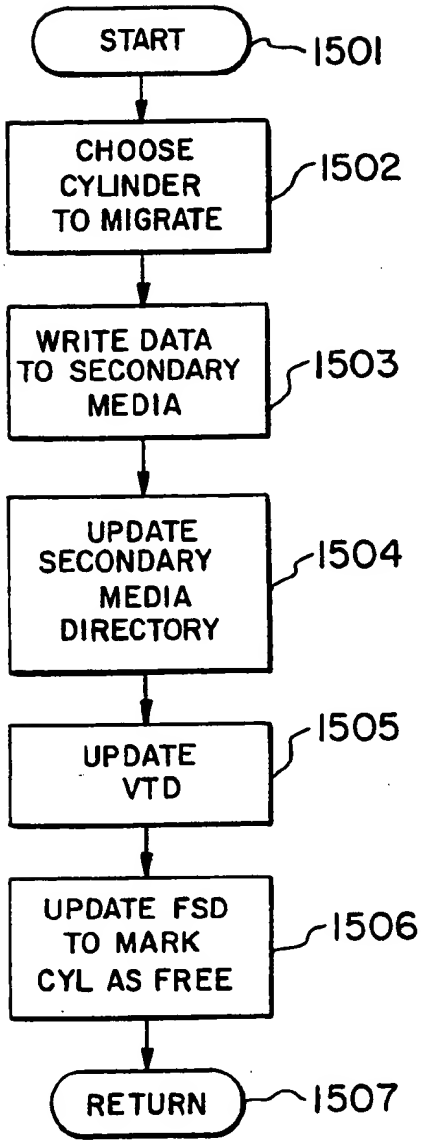

START
1501
CHOOSE CYLINDER TO MIGRATE
1502
WRITE DATA TO SECONDARY MEDIA
1503
UPDATE SECONDARY MEDIA DIRECTORY
1504
UPDATE VTD
1505
UPDATE FSD TO MARK CYL AS FREE
1506
RETURN
1507

FIG. 15.

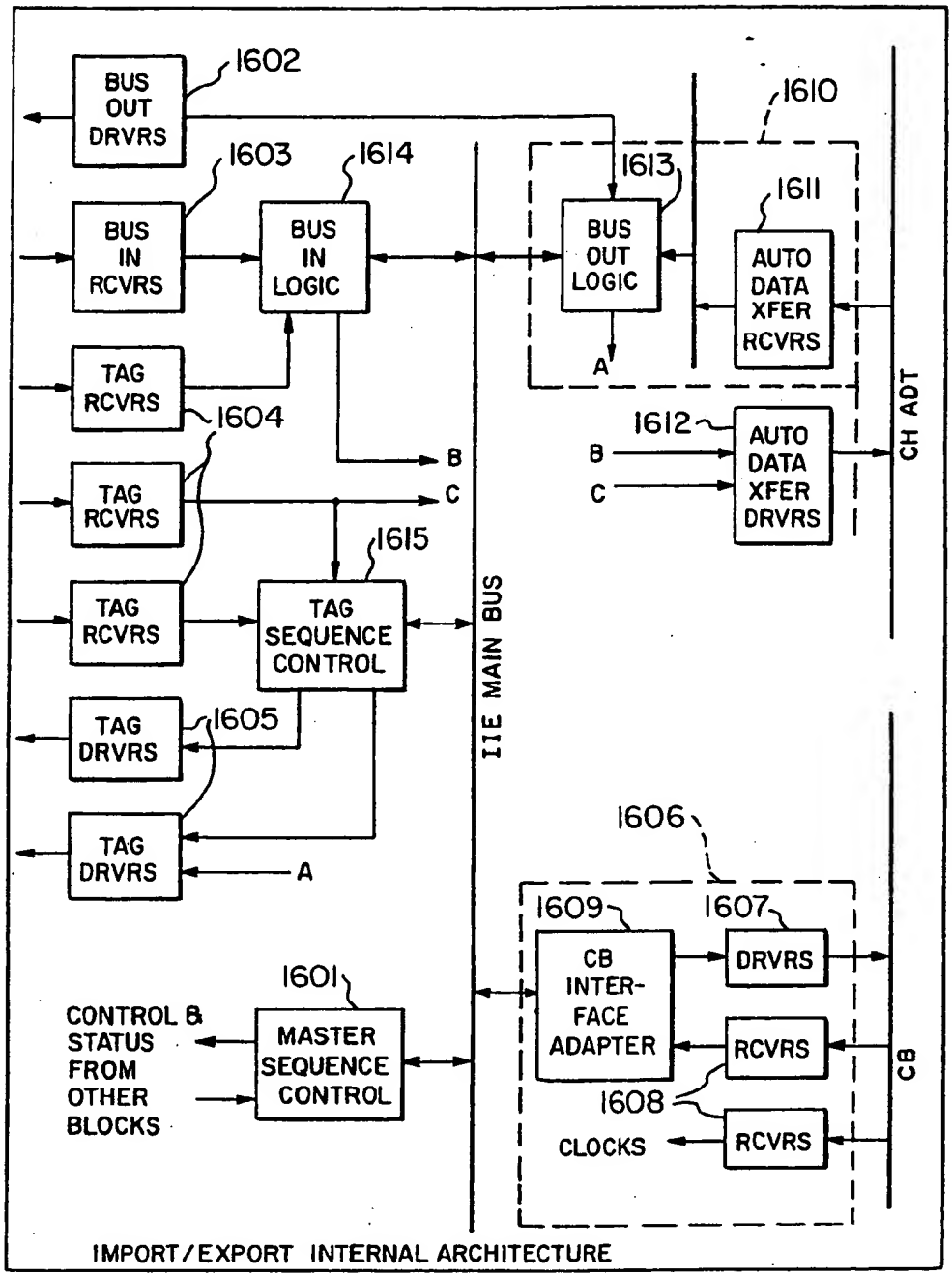

1602
BUS OUT DRVRS
1603
BUS IN RCVRS
1614
BUS IN LOGIC
1610
1613
BUS OUT LOGIC
1611
AUTO DATA XFER RCVRS
TAG RCVRS
1604
TAG RCVRS
TAG RCVRS
1615
TAG SEQUENCE CONTROL
TAG DRVRS
1605
TAG DRVRS
A
B
C
1612
AUTO DATA XFER DRVRS
CH ADT
IIE MAIN BUS
1606
1609
1607
CB INTER-FACE ADAPTER
DRVRS
RCVRS
1608
CLOCKS
RCVRS
CB
1601
CONTROL & STATUS FROM OTHER BLOCKS
MASTER SEQUENCE CONTROL
IMPORT/EXPORT INTERNAL ARCHITECTURE

FIG. 16.

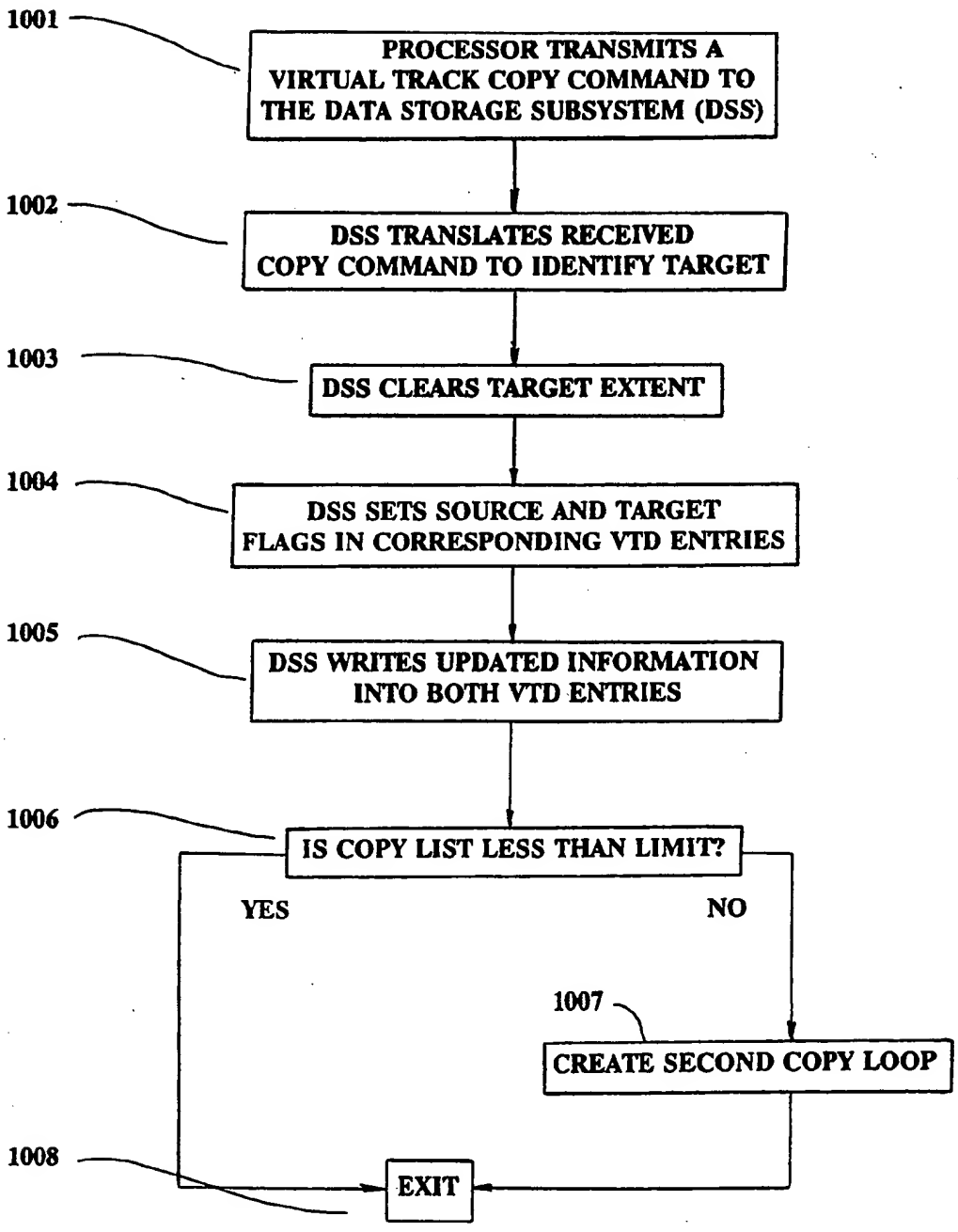

1001
PROCESSOR TRANSMITS A VIRTUAL TRACK COPY COMMAND TO THE DATA STORAGE SUBSYSTEM (DSS)
1002
DSS TRANSLATES RECEIVED COPY COMMAND TO IDENTIFY TARGET
1003
DSS CLEARS TARGET EXTENT
1004
DSS SETS SOURCE AND TARGET FLAGS IN CORRESPONDING VTD ENTRIES
1005
DSS WRITES UPDATED INFORMATION INTO BOTH VTD ENTRIES
1006
IS COPY LIST LESS THAN LIMIT?
YES
NO
1007
CREATE SECOND COPY LOOP
1008
EXIT

FIGURE 17

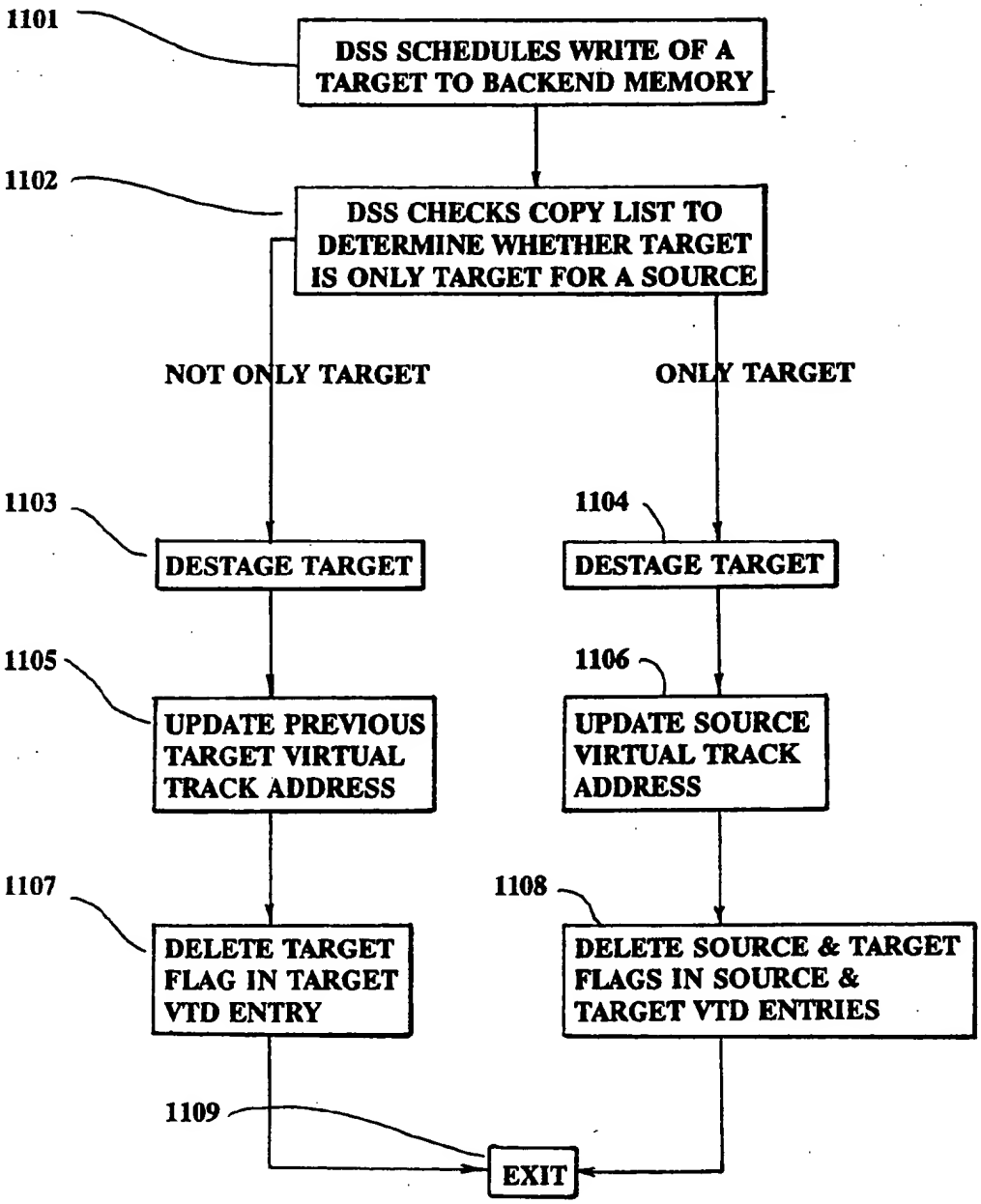

1101
DSS SCHEDULES WRITE OF A TARGET TO BACKEND MEMORY
1102
DSS CHECKS COPY LIST TO DETERMINE WHETHER TARGET IS ONLY TARGET FOR A SOURCE
NOT ONLY TARGET
ONLY TARGET
1103
DESTAGE TARGET
1104
DESTAGE TARGET
1105
UPDATE PREVIOUS TARGET VIRTUAL TRACK ADDRESS
1106
UPDATE SOURCE VIRTUAL TRACK ADDRESS
1107
DELETE TARGET FLAG IN TARGET VTD ENTRY
1108
DELETE SOURCE & TARGET FLAGS IN SOURCE & TARGET VTD ENTRIES
1109
EXIT

FIGURE 18

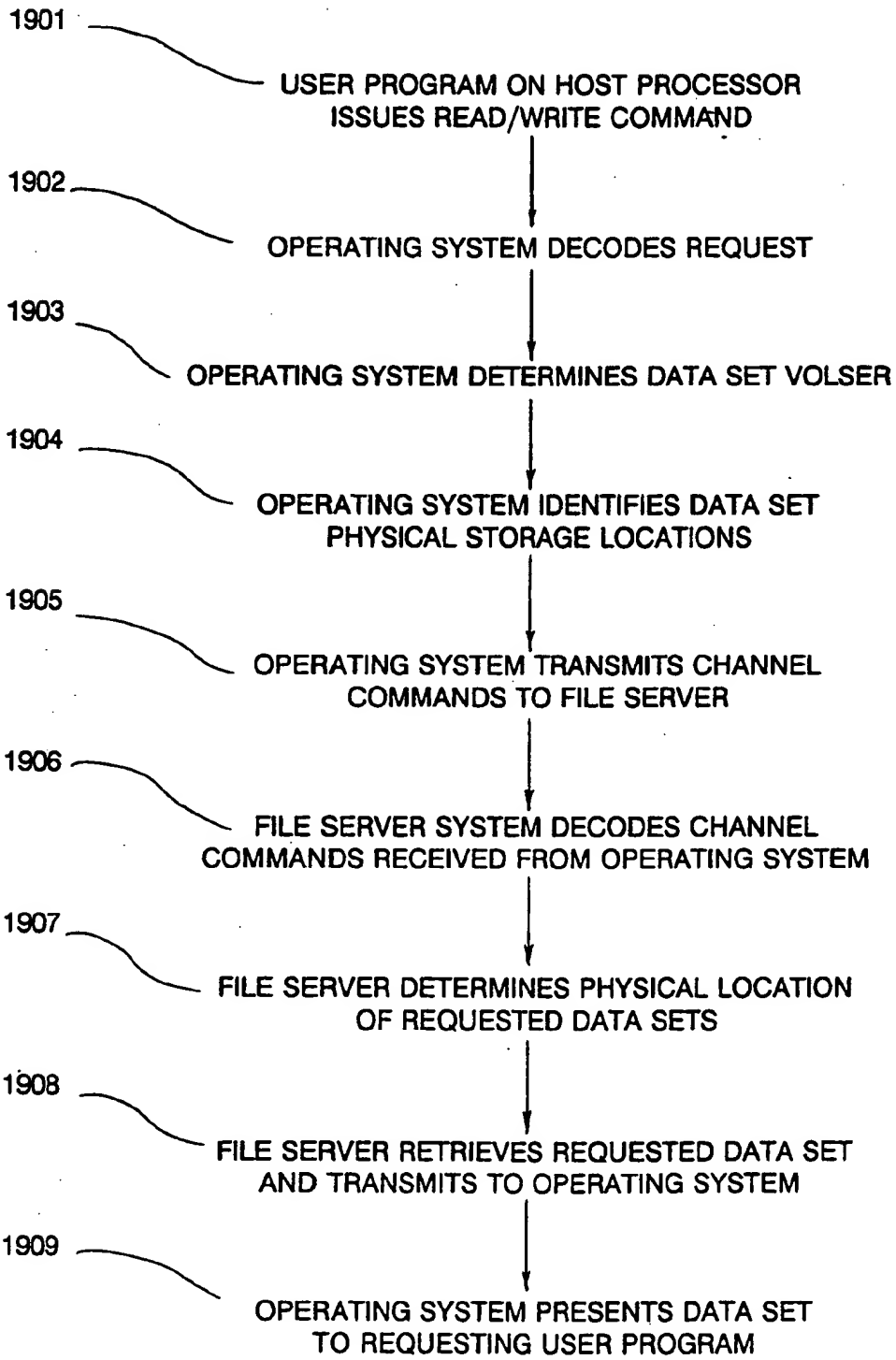

1901
USER PROGRAM ON HOST PROCESSOR ISSUES READ/WRITE COMMAND
1902
OPERATING SYSTEM DECODES REQUEST
1903
OPERATING SYSTEM DETERMINES DATA SET VOLSER
1904
OPERATING SYSTEM IDENTIFIES DATA SET PHYSICAL STORAGE LOCATIONS
1905
OPERATING SYSTEM TRANSMITS CHANNEL COMMANDS TO FILE SERVER
1906
FILE SERVER SYSTEM DECODES CHANNEL COMMANDS RECEIVED FROM OPERATING SYSTEM
1907
FILE SERVER DETERMINES PHYSICAL LOCATION OF REQUESTED DATA SETS
1908
FILE SERVER RETRIEVES REQUESTED DATA SET AND TRANSMITS TO OPERATING SYSTEM
1909
OPERATING SYSTEM PRESENTS DATA SET TO REQUESTING USER PROGRAM

FIGURE 19

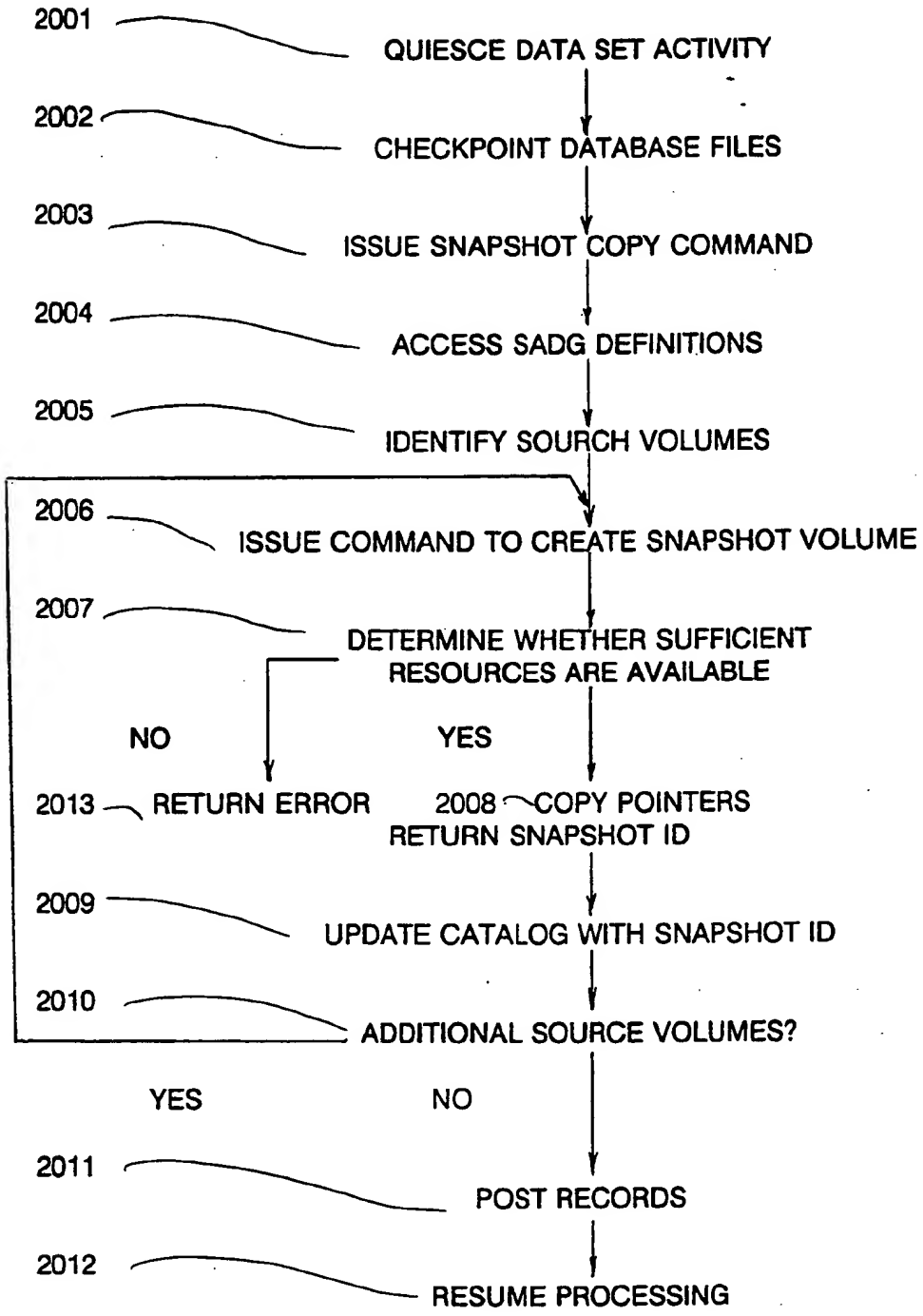

2001
QUIESCE DATA SET ACTIVITY
2002
CHECKPOINT DATABASE FILES
2003
ISSUE SNAPSHOT COPY COMMAND
2004
ACCESS SADG DEFINITIONS
2005
IDENTIFY SOURCH VOLUMES
2006
ISSUE COMMAND TO CREATE SNAPSHOT VOLUME
2007
DETERMINE WHETHER SUFFICIENT RESOURCES ARE AVAILABLE
NO
YES
2013
RETURN ERROR
2008
COPY POINTERS
RETURN SNAPSHOT ID
2009
UPDATE CATALOG WITH SNAPSHOT ID
2010
ADDITIONAL SOURCE VOLUMES?
YES
NO
2011
POST RECORDS
2012
RESUME PROCESSING

FIGURE 20

FILE SERVER HAVING SNAPSHOT APPLICATION DATA GROUPS

FIELD OF THE INVENTION

This invention relates to file servers and, in particular, to a file server system that creates and manages copies of data made external to a host processor.

PROBLEM

It is a problem in data storage subsystems to define, manage and access copies of data for a host processor. Intelligent data storage subsystems function to manage the data stored therein independent of the host processor in order to relieve the host processor of the burden of attempting to track the physical location of each data record stored in the data storage subsystem. This problem is further exacerbated by the creation of copies of data records in the data storage subsystem, especially when these copies are made independent of the host processor.

Data records are typically copied for a number of reasons, such as disaster backup, test and development of software, multiple concurrent user access, etc. The copy of a data record represents the original data record as it existed at a predefined point in time. In backup applications, copies of data records are periodically made to ensure the integrity of the data records, since backup copies are made on a regular basis. The risk of data loss is thereby minimized and restricted to the changes that have taken place since the last backup. A more complex problem is file concurrency wherein a plurality of users on a multi-user system each access a shared data record. The data storage subsystem typically makes a copy of the original data record for each of these users and then must manage this plurality of copies and the changes made thereto in order to safeguard the integrity of the original data record and properly name and manage any new or varied data records created therefrom.

The data record management problem is further complicated by the fact that in today's integrated applications, data is often more than a single data set or database. Data is more typically a set of related data sets and/or databases which must be correctly synchronized in time to ensure consistency of the set of data records. If the various data sets and/or databases are not properly synchronized, then the applications using the data will potentially make logical errors. Examples of this are the use of databases that create adjunct files consisting of indexing information, files consisting of report formats and user workspace. All of these diverse files must be properly synchronized or the entirety of the data becomes unusable.

Therefore, in data storage subsystems, copies of data records must be made accessible to the end user in a computing environment that may be homogeneous or heterogeneous. This capability must be provided without requiring modifications to the host computers or their proprietary data access methods. However, there presently exists no data storage subsystem that can define sets of data records and manage copies of them external to the host computer in a manner that is transparent to the host computer and ensures the integrity of the sets of data records.

SOLUTION

The above described problems are solved and a technical advance achieved in the field by the file server system having snapshot application data groups. This file server system appears to the data processor(s) to be a plurality of data storage devices that are directly addressable by each data processor using the native data management and access structures of the data processor. The file server system operates independent of the data processor(s) and can operate in a data storage environment that includes heterogeneous data storage media as well as heterogeneous data processors. The file server system is an intelligent data storage subsystem that defines, manages and accesses synchronized sets of data and maintains these synchronized sets of data external from the data processors' data management facilities in a manner that is completely transparent to the data processor(s). The data storage and management capability can include changing the format of the data stored to accommodate various combinations of heterogenous data processors.

This is accomplished by the use of the snapshot application data group that extends the traditional sequential data set processing concept of generation data groups. The snapshot application data groups allow the end user to define a set of data sets and/or databases that must be synchronized in time. The snapshot application data group then allows the end user to reference that set of data sets as a single entity for creation, access and deletion operations. It also provides a mechanism for managing resources consumed by the copies of the data that are created within the file server system.

In one embodiment of the present invention, a disk array data storage subsystem is used to illustrate the concept of snapshot application data groups. The-disk array data storage subsystem comprises a plurality of small form factor disk drives that are interconnected into redundancy groups, each of which contain n+m disk drives for storing n segments of data and m redundancy segments in order to safeguard the integrity of the data stored therein. Each redundancy group functions as a large form factor disk drive which image is presented to the host processor. The disk array subsystem provides internal control hardware and software to map between the virtual device image as presented to the data processor and the physical devices on which the data records are stored. This mapping consists of tables of pointers that are addressable by the data storage subsystem using the virtual image presented to the data processor and which pointers denote the physical storage location in the plurality of disk drives in a selected redundancy group that contains the desired data record.

This data storage subsystem includes a snapshot copy capability that creates copies of data records instantaneously by simply replicating pointers. In particular, when a copy of a data record is to be made, a new pointer is created, addressable at the new virtual address assigned to this copy of the data record, which pointer points to the same physical storage location in the data storage subsystem as the original data record. Therefore, the data storage subsystem simply replicates the pointer from the mapping table that points to the original set of data and assigns a new virtual address to this replicated pointer to enable the data processor to access the original data record at two different virtual addresses. A physical copy of this data record is created

only when the data processor makes changes to this data record or for backup purposes, or if the server repositions the physical data to another media, e.g. for performance reasons. Alternatively, the user may specify that the two copies must remain synchronized. In this case, the two different pointers will always reference the same physical data. This enables multiple user programs (potentially on different processors) to access a common, single physical copy of data via different logical copies.

The snapshot application data group makes use of this capability or an equivalent network functionality to make a copy of a set of data sets that are synchronized at a particular instant in time. This new copy is an instance of a snapshot application data group generation. The set of data sets may constitute a portion of a single volume or it may consist of one or more volumes whose data needs to be synchronized for recovery and data processor access purposes. This copy of the set of data sets is not accessible via the data processor's normal access methods, because the data processor retains knowledge of only the original source set of data sets on the original source virtual volumes. The file server system snapshot application data group manager retains the knowledge of the copied volumes, the user designation for these copied volumes and specific access information, including the mapping information indicative of the physical storage location on the disk drives in a redundancy group wherein the data is stored. The media used to store the data can be a disk array or any other media or combinations of media such as a disk array in combination with a backend automated magnetic tape cartridge library system, including a plurality of tape drives such that the file server system comprises a hierarchical data storage system containing multiple types of media.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 illustrates in block diagram form the overall architecture of a file server having snapshot application data groups;

FIG. 2 illustrates an overall block diagram of a disk array data storage subsystem;

FIG. 3 discloses additional details of the storage control unit of the disk array data storage subsystem;

FIG. 4 illustrates the format of a copy table;

FIG. 5 illustrates in flow diagram form copy table implementation of a data record copy operation;

FIGS. 6 and 7 illustrate, in flow diagram form, the operational steps taken to perform a data read operation;

FIG. 8 illustrates a typical free space directory used in the data storage subsystem;

FIG. 9 illustrates the format of the virtual track directory;

FIGS. 10 and 11 illustrates, in flow diagram form, the basic and enhanced free space collection processes, respectively;

FIG. 12 illustrates the format of the Logical Cylinder Directory;

FIG. 13 illustrates, in flow diagram form, the operational steps taken to perform a data write operation;

FIG. 14 illustrates a typical free space directory entry;

FIG. 15 illustrates, in flow diagram form, the migrate logical cylinder to secondary media process;

FIG. 16 illustrates additional details of an import/export control unit interface;

FIG. 17 illustrates in flow diagram form the virtual track directory implementation of a data record copy operation;

FIG. 18 illustrates in flow diagram form the steps taken to update the virtual track directory when a virtual track that was copied from another virtual track is written to the disk drives of a redundancy group;

FIG. 19 illustrates in flow diagram form the operational steps taken in a data processor file server read and write interaction; and

FIG. 20 illustrates in flow diagram form the operational steps taken by the system to create a snapshot copy of an application data group.

DETAILED DESCRIPTION OF THE DRAWING

FIG. 1 illustrates in block diagram form the overall architecture of the file server system 1 of the present invention. The file server system 1 is connected to at least one data processor 2 by a data channel 8 which functions to exchange data and control information between the data processor 2 and the file server system 1. Within the data processor 2 is resident an operating system 4 as well as a plurality of user programs 3. In addition, a catalog structure 5 is provided either in the data processor 2 or resident on a memory device 14 connected to the data processor 2. The catalog structure 5 consists of data that is used to map between the virtual address of a particular data set (also referred to as data record) and the physical location on a data storage device on which the data record is stored. The file server system 1 itself consists of a plurality of data storage devices 11-* that are used to store the data records for access by the data processor 2. As can be seen from FIG. 1, the data storage devices 11-* in the file server system 1 are divided into two groups: data storage devices 12 within the functional address space of the data processor 2, and data storage devices outside the functional address space 13 of the data processor 2. In particular, a data processor 2 can directly address N functional volumes (11-1 to 11-N) of data storage capacity while the file server system 1 can be equipped with M functional volumes 11-* of data storage capacity where $M>N$. Therefore, the directly addressable memory space available to the data processor 2 is typically less than that provided by the file server system 1. Furthermore, the directly addressable functional volumes (11-1 to 11-N) in the file server system 1 must include a plurality of functional volumes that are used as scratch functional volumes to enable the user programs 3 on the data processor 2 to read and write data thereon. Therefore, the memory address space seen by the data processor 2 consists of only a fraction of the memory capability of the file server system 1. The additional capability of the file server system 1 can therefore be used to store backup copies of the data that is directly addressable by the data processor 2.

File server system 1 operates independent of data processor 2 to manage the storage of data records external to data processor 2 and in a data storage media environment that can be heterogeneous. Multiple types of media can be included within or connected to file server system 1: rotating media data storage devices 11-*, mountable media data storage devices 10 (magnetic tape), or other file server systems 9. File server system 1 dynamically maps the data records identified by data processor 2 into physical storage locations on the data storage media within file server system 1. This mapping can include compression of the data received

from data processor 2 and even changing the format of the data for storage on the data storage media. The dynamic mapping of the data processor 2 address space is accomplished by either modifying operating system 4 to directly manage the data storage, or by including utilities on data processor 2 to correct meta data used to access the data, such as the volume table of contents, or by including a file server system utility 6 on data processor 2 to intercept the data processor input/output commands and managing the input/output in file server system 1. The file server system utility 6 is the preferred embodiment and the result of this structure is to present a uniform data storage volume image to data processor 2 in a manner that is transparent to data processor 2. The image can be that of "mountable DASD" wherein file server system 1 provides data processor 2 with access to data storage capacity that exceeds the address capacity of data processor 2. In addition, file server system 1 can store data on backend media and then stage data requested by data processor 2 to cache memory or to data storage media directly addressable by data processor 2.

Overview of Data Read/Write Operations

In order to better understand the operation of this apparatus, FIG. 19 illustrates in flow diagram form the operational steps taken by the apparatus illustrated in FIG. 1 to read and write data from the data processor 2 on the file server system 1. At step 1901, a user program 3 resident on data processor 2 issues a read or a write command. The operating system 4 at step 1902 receives the request from the user program 3 and interprets the request to determine what action is required to satisfy this request. At step 1903, the operating system 4 determines the virtual address, e.g. the volume serial number on which they reside, of the data records that are requested by the user program 3. The virtual address is obtained from the catalog structure 5. The catalog structure 5 can consist of multiple levels of information 5a that translate the virtual address provided by the user program 3 into an identification of a device or a functional volume within the file server system 1 that contains the requested data record. Additional physical location identification information 5b may be stored in the catalog 5 and maintained by file server system utility 6 to indicate a particular track and cylinder on the data storage device that contains the data record that has been requested by the user program 3. Alternatively, the data storage device itself may contain a level of catalog information that specifically points to the physical location on the data storage device of the requested data record. The physical location is determined by the operating system 4 at step 1904 by translating the virtual address of the requested data record into a physical location indicia.

In either case, at step 1905, the operating system 4 builds a channel command, or other network request for data, which consists of a command of predetermined format that is used to indicate to the file server system 1 the nature of the request from the data processor 2 and specific data relating to this request that enable the file server system 1 to satisfy the request. In the case of a data record read command, the channel command is identified through its data content as being a data record read request and additional information is provided in the channel command to identify the virtual address and the specific physical location of the requested data record. At step 1906, the operating system 4 issues the channel command to the channel controller 7 which transmits the channel command over the data link 8 to the file server system 1 to obtain the data records stored therein. At step 1906, the file server system 1 decodes the received channel command to determine the nature of the command and the physical location of the requested data record. Since this is a data record read request, the file server system 1 identifies the functional volume that is identified by the data processor channel command and uses the information contained in the channel command to either directly locate the requested data record or access further catalog information stored in the file server system 1 to identify the physical storage location in the file server system 1 that contains the requested data record. The file server system 1 retrieves the requested data record from its identified physical storage location and transmits the data and a response channel command over the data channel 8 to the data processor 2 at step 1908. At step 1909, the operating system 4, in response to the data transmitted by the file server system 1, returns the data to the user program 3 for use therein.

It is obvious in this high level functional description that the generic process described herein is presently in use but suffers from the limitation that the data processor 2 must manage all of the data records that are stored on the file server system 1. This represents a processing burden on the host processor 2 and significantly limits the amount of memory that can be accessed by the data processor 2 and used to store the data records that are accessible by the user programs 3 resident on the data processor 2.

(k) Generation Data Groups

As can be seen from the architecture of FIG. 1, the address space within the file server system 1 is of greater extent than that directly addressable by the data processor 2. As noted above, the data processor 2 can directly address N functional volumes while the file server system 1 consists of a memory capacity of M functional volumes where M>N. The use of a file server system 1 that operates independent of the data processor 2 to manage the data records stored therein is of significant benefit to the capability of the data processor 2 especially where the file server system 1 automatically manages backup data for the active functional volumes (11-1 to 11-N) that are used by the data processor 2. The user programs 3 on the data processor 2 typically make use not of a single data set but sets of data sets wherein a plurality of data sets or data records must be concurrently managed in order to ensure the integrity of the data contained therein. An example of such an application is the use of a database wherein the raw data can be stored in one functional volume 11-2 in the file server system 1 while the indexing information to cross reference the data that is stored in the database system can stored on yet another functional volume 11-3 in the file server system 1. Furthermore, preformatted standard report programs can also be stored on the file server system 1 on different functional volumes 11-1 therein. Therefore, it is obvious the sets of data sets can be scattered throughout many functional volumes in the file server system 1 and all of these data sets in the set of data sets must be temporally concurrent in order to prevent the corruption of the data that is stored therein.

46

54

(l) A well known functional capability presently found in computer systems in the generation data group. The generation data group is a data management methodology that automatically maintains a predetermined num-

ber of most recent versions of a data record. The most recent version of a data record is generally assigned a relative version number of 0 while the last most recent version of the same data record is assigned the relative version number-1. It can be seen from this methodology that a user can define the number of vintages of data that is to be saved on the file server system by simply defining the maximum relative version number that the file server system **1** can store. By setting this threshold number, the file server system **1** can automatically discard the oldest version of this data record when a new version is created by a user program **3** on the data processor **2**. This enables the file server system **1** to automatically maintain a predetermined number of prior copies of the data record. Furthermore, the file server system **1** can automatically archive a predetermined old version of the data record when a new version of the data record is created. Therefore, an example where the file server system **1** maintains three versions of a data record, when the user program **3** on the data processor **2** creates a new version of the data record, the oldest or fourth version of the data record that presently is stored on the file server system **1** can be transmitted to an archive memory that is part of the file server system **1** or an adjunct system connected thereto for automated archiving while the file server system **1** updates the version number of the remaining three copies of the data record stored therein and assigns the new version of the data record received from the data processor **2** the relative version 0 tag. In order for the file server system **1** to most expeditiously manage the plurality of versions of data records that is stored therein, a snapshot copy capability is utilized to quickly make copies of data records and to dynamically move the data records using a pointer redirection scheme that is described in detail below. Furthermore, when a data record on a functional volume is to be replicated, the address mapping must also be replicated so that the snapshot volume is identical to the original functional volume that is directly addressable by the data processor **2**.

The file server system **1** operates independent of the data processor **2** and directly manages all of the data records stored therein. By making use of data record pointer information to manage the placement of data records in the copying of these data records, the file server system **1** can quickly transfer files from the functional address space accessible by the data processor **2** to the functional address space that is outside the addressing range of the data processor **2** by simply managing the pointers that reference the physical locations on the data storage devices **11-*** contained in the file server system **1**. Therefore, without having to physically relocate the data record on the devices contained in the file server system **1**, the file server system **1** can effectively accomplish an instantaneous move or copy of a data record in a manner that appears to the data processor **2** to be a physical movement or replication of the data record from one physical device to another physical device even though the data record is not moved and the pointers thereto are simply managed by the file server system **1**. The file server system **1** maintains a dynamic mapping that is transparent to the data processor **2** and redirects a data record to a physical memory location on a data storage device contained within the file server system **1** not as a function of host processor commands but as a function of the mapping memory contained within the file server system **1**. The data processor **2** is totally unaware of the mapping and data management that takes place within the file server system and this obviates the need for the data processor **2** to attempt to manage the data records that are stored on the file server system **1**.

The preferred embodiment of the file server system **1** disclosed herein below is that of a disk array data storage subsystem **100** illustrated in FIG. **2**. The disk array data storage subsystem **100** includes an import/export control unit **208-*** that interconnects data storage subsystem **100** with additional data storage devices. In the embodiment illustrated in FIG. **2**, data storage subsystem **100** has connected thereto a archive memory **10** which in the preferred embodiment is disclosed as a tape drive subsystem that can be an automated magnetic tape cartridge library system to store and retrieve a large number of magnetic tape cartridges **10***a*, **10***b*. It is obvious that the specifics of this implementation are but one embodiment that can be used to provide the file server system capability as generally described herein. The disk array data storage subsystem **100** is selected for the preferred embodiment due to its dynamic mapping of virtual devices wherein the virtual device image presented to the data processor **2** is simply an emulation and the data is stored on physical devices in a manner known only to the data storage subsystem **100** rather than the data processor **2**.

Data Storage Subsystem Architecture

FIG. **2** illustrates in block diagram form the architecture of the preferred embodiment of the data storage subsystem **1**, including disk drive array data storage subsystem **100**. The disk drive array data storage subsystem **100** appears to the associated host processors **11-12** to be a collection of storage devices, such as large form factor disk drives with their associated storage control, since the architecture of disk drive array data storage subsystem **100** is transparent to the associated data processors **2 - 2'**. This disk drive array data storage subsystem **100** includes a plurality of disk drives (ex **122-1** to **125-r**) located in a plurality of disk drive subsets **103-1** to **103-i**. The disk drives **122-1** to **125-r** are significantly less expensive, even while providing disk drives to store redundancy information and providing disk drives for spare purposes, than the typical **14** inch form factor disk drive with an associated backup disk drive. The plurality of disk drives **122-1** to **125-r** are typically the commodity hard disk drives in the 5¼ inch form factor.

The architecture illustrated in FIG. **2** is that of a plurality of data processors **2 - 2'** interconnected via the respective plurality of data channels **21, 22-31, 32**, respectively to a data storage subsystem **100** that provides the backend data storage capacity for the data processors **2 - 2'**. This basic configuration is well known in the data processing art. The data storage subsystem **100** includes a control unit **101** that serves to interconnect the subsets of disk drives **103-1** to **103-i** and their associated drive managers **102-1** to **102-i** with the data channels **21 - 22, 31 - 32** that interconnect data storage subsystem **100** with the plurality of data processors **2 - 2'**.

Control unit **101** includes typically two cluster controls **111, 112** for redundancy purposes. Within a cluster control **111** the multipath storage director **110-0** provides a hardware interface to interconnect data channels **21, 31** to cluster control **111** contained in control unit **101**. In this respect, the multipath storage director **110-0** provides a hardware interface to the associated data channels **21, 31** and provides a multiplex function

to enable any attached data channel ex-21 from any data processor 2 to interconnect to a selected cluster control 111 within control unit 101. The cluster control 111 itself provides a pair of storage paths 201-0, 201-1 which function as an interface to a plurality of optical fiber backend channels 104. In addition, the cluster control 111 includes a data compression function as well as a data routing function that enables cluster control 111 to direct the transfer of data between a selected data channel 21 and cache memory 113, and between cache memory 113 and one of the connected optical fiber backend channels 104. Control unit 101 provides the major data storage subsystem control functions that include the creation and regulation of data redundancy groups, reconstruction of data for a failed disk drive, switching a spare disk drive in place of a failed disk drive, data redundancy generation, logical device space management, and virtual to logical device mapping. These subsystem functions are discussed in further detail below.

Disk drive manager 102-1 interconnects the plurality of commodity disk drives 122-1 to 125-r included in disk drive subset 103-1 with the plurality of optical fiber backend channels 104. Disk drive manager 102-1 includes an input/output circuit 120 that provides a hardware interface to interconnect the optical fiber backend channels 104 with the data paths 126 that serve control and drive circuits 121. Control and drive circuits 121 receive the data on conductors 126 from input/output circuit 120 and convert the form and format of these signals as required by the associated commodity disk drives in disk drive subset 103-1. In addition, control and drive circuits 121 provide a control signalling interface to transfer signals between the disk drive subset 103-1 and control unit 101.

The data that is written onto the disk drives in disk drive subset 103-1 consists of data that is transmitted from an associated data processor 2 over data channel 21 to one of cluster controls 111, 112 in control unit 101. The data is written into, for example, cluster control 111 which stores the data in cache 113. Cluster control 111 stores N physical tracks of data in cache 113 and then generates M redundancy segments for error correction purposes. Cluster control 111 then selects a subset of disk drives (122-1 to 122-n+m) to form a redundancy group to store the received data. Cluster control 111 selects an empty logical track, consisting of N+M physical tracks, in the selected redundancy group. Each of the N physical tracks of the data are written onto one of N disk drives in the selected data redundancy group. An additional M disk drives are used in the redundancy group to store the M redundancy segments. The M redundancy segments include error correction characters and data that can be used to verify the integrity of the N physical tracks that are stored on the N disk drives as well as to reconstruct one or more of the N physical tracks of the data if that physical track were lost due to a failure of the disk drive on which that physical track is stored.

Thus, data storage subsystem 100 can emulate one or more storage devices, e.g. large form factor disk drives (ex—an IBM 3380 K type of disk drive), using a plurality of smaller form factor disk drives while providing a high system reliability capability by writing the data across a plurality of the smaller form factor disk drives. A reliability improvement is also obtained by providing a pool of R spare disk drives (125-1 to 125-r) that are switchably interconnectable in place of a failed disk drive. Data reconstruction is accomplished by the use of the M redundancy segments, so that the data stored on the remaining functioning disk drives combined with the redundancy information stored in the redundancy segments can be used by control software in control unit 101 to reconstruct the data lost when one or more of the plurality of disk drives in the redundancy group fails (122-1 to 122-n+m). This arrangement provides a reliability capability similar to that obtained by disk shadowing arrangements at a significantly reduced cost over such an arrangement.

Disk Drive

Each of the disk drives 122-1 to 125-r in disk drive subset 103-1 can be considered a disk subsystem that consists of a disk drive mechanism and its surrounding control and interface circuitry. The disk drive consists of a commodity disk drive which is a commercially available hard disk drive of the type that typically is used in personal computers. A control processor associated with the disk drive has control responsibility for the entire disk drive and monitors all information routed over the various serial data channels that connect each disk drive 122-1 to 125-r to control and drive circuits 121. Any data transmitted to the disk drive over these channels is stored in a corresponding interface buffer which is connected via an associated serial data channel to a corresponding serial/parallel converter circuit. A disk controller is also provided in each disk drive to implement the low level electrical interface required by the commodity disk drive. The commodity disk drive has an interface which must be interfaced with control and drive circuits 121. The disk controller provides this function. Disk controller provides serialization and deserialization of data, CRC/ECC generation, checking and correction and NRZ data encoding. The addressing information such as the head select and other type of control signals are provided by control and drive circuits 121 to commodity disk drive 122-1. This communication path is also provided for diagnostic and control purposes. For example, control and drive circuits 121 can power a commodity disk drive down when the disk drive is in the standby mode. In this fashion, commodity disk drive remains in an idle state until it is selected by control and drive circuits 121.

Control Unit

FIG. 3 illustrates in block diagram form additional details of cluster control 111. Multipath storage director 110 includes a plurality of channel interface units 201-0 to 201-7, each of which terminates a corresponding pair of data channels 21, 31. The control and data signals received by the corresponding channel interface unit 201-0 are output on either of the corresponding control and data buses 206-C, 206-D, or 207-C, 207-D, respectively, to either storage path 200-0 or storage path 200-1. Thus, as can be seen from the structure of the cluster control 111 illustrated in FIG. 3, there is a significant amount of symmetry contained therein. Storage path 200-0 is identical to storage path 200-1 and only one of these is described herein. The multipath storage director 110 uses two sets of data and control busses 206-D, C and 207-D, C to interconnect each channel interface unit 201-0 to 201-7 with both storage path 200-0 and 200-1 so that the corresponding data channel 21 from the associated data processor 2 can be switched via either storage path 200-0 or 200-1 to the plurality of optical fiber backend channels 104. Within storage path 200-0 is contained a processor 204-0 that regulates the operation of storage path 200-0. In addition, an optical

device interface 205-0 is provided to convert between the optical fiber signalling format of optical fiber backend channels 104 and the metallic conductors contained within storage path 200-0. Channel interface control 202-0 operates under control of processor 204-0 to control the flow of data to and from cache memory 113 and the one of channel interface units 201 that is presently active within storage path 200-0. The channel interface control 202-0 includes a cyclic redundancy check (CRC) generator/checker to generate and check the CRC bytes for the received data. The channel interface circuit 202-0 also includes a buffer that compensates for speed mismatch between the data transmission rate of the data channel 21 and the available data transfer capability of the cache memory 113. The data that is received by the channel interface control circuit 202-0 from a corresponding channel interface circuit 201 is forwarded to the cache memory 113 via channel data compression circuit 203-0. The channel data compression circuit 203-0 provides the necessary hardware and microcode to perform compression of the channel data for the control unit 101 on a data write from the data processor 2. It also performs the necessary decompression operation for control unit 101 on a data read operation by the data processor 2.

As can be seen from the architecture illustrated in FIG. 2, all data transfers between a data processor 2 and a redundancy group in the disk drive subsets 103 are routed through cache memory 113. Control of cache memory 113 is provided in control unit 101 by processor 204-0. The functions provided by processor 204-0 include initialization of the cache directory and other cache data structures, cache directory searching and management, cache space management, cache performance improvement algorithms as well as other cache control functions. In addition, processor 204-0 creates the redundancy groups from the disk drives in disk drive subsets 103 and maintains records of the status of those devices. Processor 204-0 also causes the redundancy data across the N data disks in a redundancy group to be generated within cache memory 113 and writes the M segments of redundancy data onto the M redundancy disks in the redundancy group. The functional software in processor 204-0 also manages the mappings from virtual to logical and from logical to physical devices. The tables that describe this mapping are updated, maintained, backed-up-and-occasionally recovered by this functional software on processor 204-0. The free space collection function is also performed by processor 204-0 as well as management and scheduling of the optical fiber backend channels 104. Many of these above functions are well known in the data processing art and are not described in any detail herein.

Data Compression Capabilities

Data stored on disks and tapes or transferred over communication links in a computer system generally contains significant redundancy. Data compression algorithms improve the efficiency with which data is stored or transmitted by reducing the amount of redundant data. A compression algorithm takes a source text as input an produces a corresponding compressed text. An expansion algorithm takes the compressed text as input and produces the original source text as an output. There are four types of redundancy that are typically found in a data file. The first type of redundancy is character distribution redundancy. In a typical character string, some characters are used more frequently than others. Specifically, the eight bit ASCII representations used to encode characters are not all used in a typical character string. Nearly three fourths of the possible 256 characters representations may not be used in a specific file. Consequently, nearly two bits of each eight bit representation could be removed without affecting the data content of the character string. This is a twenty-five percent savings in space and encoding time. The second type of redundancy is character repetition redundancy. When a string of repetitions of a single character occurs, the message can be encoded more compactly than by just repeating the character symbol. The character repetition strings are infrequent in text but fairly common in formatted business files where unused space is very common. In addition, graphical images contain a significant amount of character repetition redundancy. A third form of redundancy consists of high usage patterns. Certain sequences of characters appear with relatively high frequency in a particular data file and therefore can be represented with relatively fewer bits for a net savings in data storage space and string encoding time. Thus, frequently occurring patterns are encoded using fewer bits while infrequently occurring patterns are encoded using more bits. The fourth type of redundancy is positional redundancy. If certain characters appear consistently at a predictable place in each block of data, then the characters are at least partially redundant. An example of positional redundancy are charts and pictures.

The most popular method of data compression is Huffman type coding which translates fixed sized pieces of input data into variable length symbols. The Huffman encoding procedure assigns codes to input symbols such that each code length is proportional to the probability of the symbol occurring in the data. In normal use, the size of the input symbols is limited by the size of the translation table needed for compression. That is, a table is needed that lists each input symbol and its corresponding code. A second problem with Huffman encoding is the complexity of the decompression process. The length of each code to be interpreted for decompression is not known until the first few bits are interpreted. An improvement over Huffman coding is an adaptive compression algorithm such as the Lempel-Ziv category of algorithms that converts variable length strings of input symbols into fixed length codes. This form of data compression is effective at exploiting character frequency redundancy, character repetition redundancy, and high usage pattern redundancy but is not generally effective on positional redundancy. This algorithm is adaptive in the sense that it starts each field with an empty table of symbol strings and builds the table during both the compression and decompression processes. These are one PASS procedures that require no prior information about the input data statistics and execute in time proportional to the length of the message.

The length of a compressed image for a given message is unpredictable because it depends on the content of the message. There is no assurance prior to data compression that a message will compress at all; in some cases it may even expand. Therefore, the space allocated for the compressed image must be at least as big as the space allocated for the original message. In addition, an update to a data record that alters just a few characters of the data record can change the compressed size and may result in a required change in allocation even for a minor update. Therefore, the above-described

process used by data storage subsystem **100** to perform modifications to data records overcomes this minor update impact on compressed data, since a modified data record is always written to an empty logical cylinder and the old version of the data record is flagged as obsolete.

Data Compaction

As a data record is received from data processor **2** by channel interface control **202-0**, and buffered therein, processor **204-0** deletes all gaps between fields in the received count key data record. The virtual device and virtual cylinder addresses are extracted from the count key data format data record and used to create an entry in the virtual cylinder directory stored in cache memory **113**. The data fields of the received data record are forwarded to channel data compression circuit **203-0** for compression and temporary storage in cache memory **113**. Thus, all that is stored in the redundancy groups are logical cylinders of compressed data in fixed block architecture format since the headers, gaps and received space in the received count key data are deleted. A further compaction process is the creation of null virtual tracks. Each time data processor **2** creates a new instance of a data file, a predetermined data file extent is reserved by data processor **2**. Channel interface control **202-0** and processor **204-0** eliminate the need to reserve this unused memory space by simply creating a series of null entries in the virtual track directory; no data is written to a redundancy group.

Adaptive Data Compression Function

The adaptive data compression apparatus **203-0** is located within control unit **101**, which is interposed between a plurality of host processor channel interface units **201** and cache memory **113**. The adaptive data compression apparatus **203-0** functions to efficiently compress the records of a user data file received from the data processor **2** into a bit oriented compressed format for storage in cache memory **113** and disk drives **122**. The data compression apparatus **203-0** divides each record of an incoming stream of data records into predetermined sized segments, each of which is compressed independently without reference to any other segment in the stream of data records. The data compression apparatus **203-0** concurrently uses a plurality of data compression algorithms to adapt the data compression operation to the particular data stored in the user data record. A cyclic redundancy check circuit is used to compute a predetermined length CRC code from all of the incoming user data bytes before they are compressed. The computed CRC code is appended to the end of the compressed data block.

The data compression apparatus **203-0** operates by converting bytes and strings of bytes into shorter bit string codes called reference values. The reference values replace the bytes and strings of bytes when recorded on the disk drives **122**. The byte strings have two forms, a run length form for characters that are repeated three or more times, and a string form that recognizes character patterns of two or more characters. Two variables are used to indicate the maximum and minimum byte values in a particular segment.

Strings of two or more bytes are compressed by assigning a reference value to each defined string using an adaptive data compression algorithm. Subsequent occurrences of that string are replaced by its string reference value. Strings are constructed a character at a time, where a previously defined string plus the next user data byte defines a new string and is assigned the next previously undefined reference value. Thus, strings become longer and data compression more efficient as more user data bytes in the segment are examined. However, as more strings are defined, greater length reference values are needed to uniquely identify a string, reducing the efficiency of the compression process. This factor makes it desirable to divide a data record into segments which are compressed independently. String definition occurs by combining the last used reference value with the next user data byte from the input data stream, then searching to see if this string has been previously defined. If it has, the next byte is concatenated to this new byte string reference value and a search is again conducted to see if this extended byte string has been previously defined as a string. This sequence is continued until a string is located that has not been previously defined. The last used defined string reference value is put in the compressed output data stream and the next previously undefined reference value is assigned to define the last string that was not found. The search procedure is initiated over again starting with the most recently used user data byte.

Runs of three or more repeated bytes are encoded using a predetermined set of reserved reference values to indicate that the preceding character was repeated the number of times specified by the repeat code. The immediately preceding character is not re-included in the repeat count. Run length encoding takes precedence over string data compression. Run length encoding, single byte compression, and string data compression are intermixed in any compressed segment within the user data record.

If the size of a compressed segment in bytes is larger than its size before compression then the segment is not compressed and is recorded in uncompressed format.

Import/Export Control Unit Interface

The file server system **1** can include several types of media and these can be disk drives **12*-*** of differing format and capacity or even different media, such as tape **10**, connected to data storage subsystem **100** via an import/export interface unit **208-***. Data records can be migrated or copied, either automatically within file server system **1** or by command from data processor **2** between data storage subsystem **100**, a similar subsystem at another location, and/or tape drive control unit **10**. In the particular embodiment disclosed herein, a tape drive control unit **10** is described, but it is understood that import/export interface unit **208-*** can be connected to any data storage media or to a network that can be used to access data storage media. This data storage media can be collocated with data storage subsystem **100** or can be spatially disjunct therefrom as in the case of a geographically distant secure archive facility.

FIG. **16** illustrates in block diagram form additional details of the tape drive control unit interface **208-1** which is connected via data channel **20** to tape drive control unit **10** which interconnects the data channel **20** with a plurality of tape drives. Tape drive control unit interface **208** is similar in structure to a data channel interface circuit **201** and functions like a host channel interface so that the tape drive control unit **10** believes that data channel **20** is a normal IBM OEMI type channel. FIG. **16** illustrates the master sequence control **1601** which is the main functional control of the tape drive control unit interface circuit **208**. All other control function in the tape drive control unit interface circuit **208** are slaves to the master sequence control circuit

1601. Master sequence control 1601 recognizes and responds to sequences of events that occur on the data channel 20 for those initiated by elements within control cluster 111. Master sequence control 1601 contains a microsequencer, instruction memory, bus source and destination decode registers and various other registers as are well known in the art. A plurality of bus input receivers 1603 and bus output drivers 1602 and tag receivers 1604 and drivers 1605 are provided to transmit tag or bus signals to the tape drive control unit 10. These transmitters and receivers conform to the requirements set in the IBM OEMI specification so that normal IBM channels can be used to connect data storage subsystem 100 with a conventional tape drive control unit 10. The details of these drivers and receivers are well known in the art and are not disclosed in any detail herein. Control signals and data from processor 204 in cluster control 111 are received in the tape drive control unit interface 208-1 through the control bus interface 1606 which includes a plurality of drivers and receivers 1607, 1608 and an interface adapter 1609 which contains FIFOs to buffer the data transmitted between the main bus of the tape drive control unit interface circuit 208-1 and data and control busses 206-D, 206-C, respectively. Furthermore, automatic data transfer interface 1610 is used to transfer data between the tape interface drivers and receivers 1602, 1603 and cache memory 113 on bus CH ADT via receivers and transmitters 1611, 1612. Thus, the function of tape drive control unit interface circuit 208-1 is similar to that of channel interface circuits 201 and serve to interconnect a standard tape drive control 10 via data channel 20 to data storage subsystem 100 to exchange data and control information therebetween.

Dynamic Virtual Device to Logical Device Mapping

With respect to data transfer operations, all data transfers go through cache memory 113. Therefore, front end or channel transfer operations are completely independent of backend or device transfer operations. In this system, staging operations are similar to staging in other cached disk subsystems but destaging transfers are collected into groups for bulk transfers. In addition, this data storage subsystem 100 simultaneously performs free space collection, mapping table backup, and error recovery as background processes. Because of the complete front end/backend separation, the data storage subsystem 100 is liberated from the exacting processor timing dependencies of previous Count Key Data disk subsystems. The subsystem is free to dedicate its processing resources to increasing performance through more intelligent scheduling and data transfer control.

The disk drive array data storage subsystem 100 consists of three abstract layers: virtual, logical and physical. The virtual layer functions as a conventional large form factor disk drive memory. The logical layer functions as an array of storage units that are grouped into a plurality of redundancy groups (ex 122-1 to 122-n+m), each containing N+M disk drives to store N physical tracks of data and M physical tracks of redundancy information for each logical track. The physical layer functions as a plurality of individual small form factor disk drives, or other media. The data storage management system operates to effectuate the mapping of data among these abstract layers and to control the allocation and management of the actual space on the physical devices. These data storage management functions are performed in a manner that renders the operation of the disk drive array data storage subsystem 100 transparent to the data processors (2 - 2').

A redundancy group consists of N+M disk drives. The redundancy group is also called a logical volume or a logical device. Within each logical device there are a plurality of logical tracks, each of which is the set of all physical tracks in the redundancy group which have the same physical track address. These logical tracks are also organized into logical cylinders, each of which is the collection of all logical tracks within a redundancy group which can be accessed at a common logical actuator position. Disk drive array data storage subsystem 100 appears to the host processor to be a collection of large form factor disk drives, each of which contains a predetermined number of tracks of a predetermined size called a virtual track. Therefore, when the data processor 2 transmits data over the data channel 21 to the data storage subsystem 100, the data is transmitted in the form of the individual records of a virtual track. In order to render the operation of the disk drive array data storage subsystem 100 transparent to the data processor 2, the received data is stored on the actual physical disk drives (122-1 to 122-n+m) in the form of virtual track instances which reflect the capacity of a track on the large form factor disk drive that is emulated by data storage subsystem 100. Although a virtual track instance may spill over from one physical track to the next physical track, a virtual track instance is not permitted to spill over from one logical cylinder to another. This is done in order to simplify the management of the memory space.

When a virtual track is modified by the data processor 2, the updated instance of the virtual track is not rewritten in data storage subsystem 100 at its original location but is instead written to a new logical cylinder and the previous instance of the virtual track is marked obsolete. Therefore, over time a logical cylinder becomes riddled with "holes" of obsolete data known as free space. In order to create whole free logical cylinders, virtual track instances that are still valid and located among fragmented free space within a logical cylinder are relocated within the disk drive array data storage subsystem 100 in order to create entirely free logical cylinders. In order to evenly distribute data transfer activity, the tracks of each virtual device are scattered as uniformly as possible among the logical devices in the disk drive array data storage subsystem 100. In addition, virtual track instances are padded out if necessary to fit into an integral number of physical device sectors. This is to insure that each virtual track instance starts on a sector boundary of the physical device.

Virtual Track Directory

FIG. 9 illustrates the format of the virtual track directory 900 that is contained within cache memory 113. The virtual track directory 900 consists of the tables that map the virtual addresses as presented by data processor 2 to the logical drive addresses that is used by control unit 101. There is another mapping that takes place within control unit 101 and this is the logical to physical mapping to translate the logical address defined by the virtual track directory 900 into the exact physical location of the particular disk drive or secondary media that contains data identified by the data processor 2. The virtual track directory 900 is made up of two parts: the virtual track directory pointers 901 in the virtual device table 902 and the virtual track directory 903 itself. The virtual track directory 903 is not contigu-

ous in cache memory **113** but is scattered about the physical extent of cache memory **113** in predefined segments (ex **903-1**). Each segment **903-1** has a virtual to logical mapping for a predetermined number of cylinders, for example 64 cylinders worth of IBM 3380 type DASD tracks. In the virtual device table **902**, there are pointers to as many of these segments **903** as needed to emulate the number of cylinders configured for each of the virtual devices defined by data processor **2**. The virtual track directory **900** is created by control unit **101** at the virtual device configuration time. When a virtual volume is configured, the number of cylinders in that volume is defined by the data processor **2**. A segment **903-1** or a plurality of segments of volatile cache memory **113** are allocated to this virtual volume defined by data processor **2** and the virtual device table **902** is updated with the pointers to identify these segments **903** contained within cache memory **113**. Each segment **903** is initialized with no pointers to indicate that the virtual tracks contained on this virtual volume have not yet been written. Each entry **905** in the virtual device table is for a single virtual track and is addressed by the virtual track address. As shown in FIG. **9**, each entry **905** is 64 bits long. The entry **905** contents are as follows starting with the high order bits:

Bits 63: Migrated to Secondary Media Flag.

Bit 62: Source Flag.

Bit 61: Target Flag.

Bits 60-57: Logical volume number. This entry corresponds to the logical volume table described above.

Bits 56-46: Logical cylinder address. This data entry is identical to the physical cylinder number.

Bits 45-31: Sector offset. This entry is the offset to the start of the virtual track instance in the logical cylinder, not including the parity track sectors. These sectors are typically contained 512 bytes.

Bits 30-24: Virtual track instance size. This entry notes the number of sectors that are required to store this virtual track instance.

Bits 23-0: Virtual Track Access Counter. This entry contains a running count of the number of times the Virtual Track has been staged.

If the Migrated to Secondary Media Flag is clear, the rest of the entry contains the fields described above. If the Migrated to Secondary Media Flag is set, the Logical Cylinder containing the Virtual Track has been migrated to the Secondary Media and the rest of the entry contains a pointer to a Secondary Media Directory.

Secondary Media Directory

The Secondary Media Directory contains pointers to all the data that has been migrated and is no longer resident on the DASD contained in the subsystem, The Secondary Media Directory also contains a Retrieving flag for each Logical Cylinder indicating that the data is in the process of being retrieved, The Secondary Media Directory is kept in cache and is backed up along with the Virtual Track Directory to allow recovery in the event of a cache failure.

Logical Cylinder Directory

FIG. **12** illustrates the format of the Logical Cylinder Directory, Each Logical Cylinder that is written contains in its last few sectors a Logical Cylinder Directory (LCD). The LCD is an index to the data in the Logical Cylinder and is used primarily by Free Space Collection to determine which Virtual Tracks Instances in the Logical Cylinder are valid and need to be collected. FIG. **12** shows the LCD in graphic form, The Logical Cylinder Sequence Number uniquely identifies the Logical Cylinder and the sequence in which the Logical Cylinders were created. It is used primarily during Mapping Table Recovery operations. The Logical Address is used as a confirmation of the Cylinders location for data integrity considerations. The LCD Entry count is the number of Virtual Track Instances contained in the Logical Cylinder and is used when scanning the LCD Entries, The Logical Cylinder Collection History contains when the cylinder was created, whether it was created from Updated Virtual Track Instances or was created from data collected from another cylinder, and if it was created from collected data, what was the nature of the collected data. The LCD Entry itself contains the identifier of the virtual track and the identifier of the relative sector within the logical cylinder in which the virtual track instance begins.

Free Space Directory

The storage control also includes a free space directory (FIG. **8**) which is a list of all of the logical cylinders in the disk drive array data storage subsystem **100** ordered by logical device. Each logical device is cataloged in two lists called the free space list and the free cylinder list for the logical device; each list entry represents a logical cylinder and indicates the amount of free space that this logical cylinder presently contains. This free space directory contains a positional entry for each logical cylinder; each entry includes both forward and backward pointers for the doubly linked free space list for its logical device and the number of free sectors contained in the logical cylinder. Each of these pointers points either to another entry in the free space list for its logical device or is null. In addition to the pointers and free sector count, the Free Space Directory also contains entries that do not relate to Free Space, but relate to the Logical Cylinder. There is a flag byte known as the Logical Cylinder Table (LCT) which contains, among other flags, a Collected Flag and an Archive Flag. The Collected Flag is set when the logical cylinder contains data that was collected from another cylinder. The Archive Flag is set when the logical cylinder contains data that was collected from a logical cylinder which had its Collected Flag set. If either one of these flags is set, the Access Counter and the Last Access Date/Time is valid. The Creation Time/Date is valid for any cylinder that is not free.

The collection of free space is a background process that is implemented in the disk drive array data storage subsystem **100**. The free space collection process makes use of the logical cylinder directory, which is a list contained in the last few sectors of each logical cylinder, indicative of the contents of that logical cylinder. The logical cylinder directory contains an entry for each virtual track instance contained within the logical cylinder. The entry for each virtual track instance contains the identifier of the virtual track instance and the identifier of the relative sector within the logical cylinder in which the virtual track instance begins. From this directory and the virtual track directory, the free space collection process can determine which virtual track instances are still current in this logical cylinder and therefore need to be moved to another location to make the logical cylinder available for writing new data.

Mapping Tables

It is necessary to accurately record the location of all data within the disk drive array data storage subsystem **100** since the data received from the data processors **2** - **2'** is mapped from its address in the virtual space to a

physical location in the subsystem in a dynamic fashion. A virtual track directory is maintained to recall the location of the present instance of each virtual track in disk drive array data storage subsystem **100**. Changes to the virtual track directory are journaled to a non-volatile store and are backed up with fuzzy image copies to safeguard the mapping data. The virtual track directory **3** consists of an entry **300** (FIG. **9**) for each virtual track which the associated data processor **2** can address. The virtual track directory entry **300** also contains data **307** indicative of the length of the virtual track instance in sectors. The virtual track directory **3** is stored in noncontiguous pieces of the cache memory **113** and is addressed indirectly through pointers in a virtual device table. The virtual track directory **3** is updated whenever a new virtual track instance is written to the disk drives.

The storage control also includes a free space directory **800** (FIG. **8**) which is a list of all of the logical cylinders in the disk drive array data storage subsystem **100** ordered by logical device. Each logical device is cataloged in a list called a free space list **801** for the logical device; each list entry represents a logical cylinder and indicates the amount of free space that this logical cylinder presently contains. This free space directory contains a positional entry for each logical cylinder; each entry includes both forward **802** and backward **803** pointers for the doubly linked free space list **801** for its logical device and the number of free sectors contained in the logical cylinder. Each of these pointers **802, 803** points either to another entry in the free space list **801** for its logical device or is null. The collection of free space is a background process that is implemented in the disk drive array data storage subsystem **100**. The free space collection process makes use of the logical cylinder directory, which is a list contained in the last few sectors of each logical cylinder indicative of the contents of that logical cylinder. The logical cylinder directory contains an entry for each virtual track instance contained within the logical cylinder. The entry for each virtual track instance contains the identifier of the virtual track instance and the identifier of the relative sector within the logical cylinder in which the virtual track instance begins. From this directory and the virtual track directory, the free space collection process can determine which virtual track instances are still current in this logical cylinder and therefore need to be moved to another location to make the logical cylinder available for writing new data.

Data Move/Copy Operation

The data record move/copy operation instantaneously relocates or creates a second instance of a selected data record by merely generating a new pointer to reference the same physical memory location as the original reference pointer in the virtual track directory. In this fashion, by simply generating a new pointer referencing the same physical memory space, the data record can be moved/copied.

This apparatus instantaneously moves the original data record without the time penalty of having to download the data record to the cache memory **113** and write the data record to a new physical memory location. For the purpose of enabling a program to simply access the data record at a different virtual address, the use of this mechanism provides a significant time advantage. A physical copy of the original data record can later be written as a background process to a second memory location, if so desired. Alternatively, when one of the programs that can access the data record writes data to or modifies the data record in any way, the modified copy of a portion of the original data record is written to a new physical memory location and the corresponding address pointers are changed to reflect the new location of this rewritten portion of the data record.

In this fashion, a data record can be instantaneously moved/copied by simply creating a new memory pointer and the actual physical copying of the data record can take place either as a background process or incrementally as necessary when each virtual track of the data record is modified by one of the programs that accesses the data record. This data record copy operation can be implemented in a number of different ways. A first method of manipulating memory pointers is to use a lookaside copy table which functions as a map to be used by the data storage subsystem **100** to list all the data records that are accessible by more than one virtual address. A second method of manipulating data record pointers is to provide additional data in the virtual track directory **3** in order to record the copy status of each data record therein. These two methods each have advantages and disadvantages in the implementation of the data record pointer management function and are disclosed herein as implementations illustrative of the concept of this invention.

Copy Table Implementation

Each entry **300** in the Virtual Track Directory (VTD) **3** contains two flags associated with the Copy/-Move function. The "Source" flag **306** is set whenever a Virtual Track Instance at this Virtual Track Address has been the origin of a copy or move. The Virtual Track Instance pointed to by this entry **300** is not necessarily the Source, but the Virtual Track Instance contains this Virtual Address. If the Source flag **306** is set, there is at least one entry in the Copy Table **400** (FIG. **4**) for this Virtual Address. The "Target" flag **303** is set whenever a Virtual Track Instance contains data that has been the destination of a copy or move. If the Target flag **303** is set, the Virtual Address in the Virtual Track Instance that is pointed to is not that of the Virtual Track Directory Entry **300**.

The format of the Copy Table **400** is illustrated graphically in FIG. **4**. The preferred implementation is to have a separate lookaside Copy Table **400** for each Logical Device so that there is a Copy Table head **401** and tail **402** pointer associated with each Logical Device; however, the copy table **400** could just as easily be implemented as a single table for the entire data storage subsystem **100**. In either case, the changes to the copy table **400** are journaled as noted above for the virtual track directory. The copy table is ordered such that the sources **4*0** are in ascending Logical Address order. The copy table **400** is a singly linked list of Sources **4*0** where each Source (such as **410**) is the head of a linked list of Targets **411,412**. The Source Entry **410** contains the following data:

Logical Address (VTD Entry Copy)
Virtual Address
Next Source Pointer (NULL if last Source in list)
Target Pointer
The Target Entry 411 contains the following data:
Virtual Address
Next Target Pointer (NULL if last Target in list)

-continued

```
Update Count Fields Flag
```

Snapshot Copy Operation Using Copy Table

FIG. **5** illustrates in flow diagram form the operational steps taken by data storage subsystem **100** to produce a copy of a virtual track instance (also referred to as data record) using the copy table implementation of the snapshot copy operation. When data processor **2** transmits a data copy request to data storage subsystem **100** over data link **21** at step **501**, the control software in processor **204-0** for example translates the received data copy request into an identification of a particular virtual track directory entry **300** stored in cache memory **113** at step **502**. Processor **204-0** in data storage subsystem **100** verifies at step **503** that the extents are defined, the same length and do not overlap. The cache management software ensures at step **504** that all the tracks in this target extent are cleared and available for the copy operation. Processor **204-0** reads the virtual track directory entry **300** and creates at step **505** a copy of this entry to be used as the virtual track directory entry for the target virtual track. At step **506** processor **204-0** sets the source **306** and target **303** flags respectively in the original and copied virtual track directory entries. Processor **204-0** then writes at step **507** the updated virtual track directory entry for the source virtual track back into the virtual track directory **3** as well as the new virtual track directory entry for the target virtual track into the virtual track directory **3**. At step **508**, a determination is made whether the source virtual track is already listed in copy table **400**. If the source virtual track is not already a source or a target virtual track in copy table **400**, then both the source entry and a target entry are created by processor **204-0** and written at step **509** into copy table **400** in the form noted above with respect to FIG. **4**. If the source data record was already marked as a source or a target data record in copy table **400**, then copy table **400** is scanned at step **510** in order to locate this entry and the target entry is added to this linked list to create a new target for this source data record. A more specific recitation of this process is illustrated in the following pseudo code:

```
Read VTD Entry for Source
Set Source Flag in VTD Entry
Write updated VTD Entry for Source back to VTD
Set Target Flag in a copy of the VTD Entry
Read VTD entry for Target
Increase Free Space for Cylinder pointed to by
 old VTD entry
Reorder Free List, if necessary
Write updated VTD Entry to Target location in
 VTD
Create Target Entry for the Copy Table
 Move the Update Count Fields Flag from the
  command to the Target Entry
If Source is NOT already a Source or a Target
    Create Source Entry for Copy Table
    Link Source into proper location in Copy
     Table
    Link Target Entry to Source Entry in Copy
     Table
Elseif (New Source was already marked as
 Source)
    Scan Source List to find Source in Copy
     Table
    If find Source
        Link Target to Last Target in this
         Source's Target List
    Else (scanned to end of Source List)
```

-continued

```
        Create Source Entry for Copy Table
        Link Source into proper location in
         source list
        Link Target Entry to Source Entry in
         Copy Table
    Endif
Else (New Source was already in Copy Table as
 Target)
    Scan Source List to find Logical Address of
     Target
    Link Target Entry to Last Target in this
     Source's Target List
Endif
Journal the changes to the VTD and to the Copy
 Table
```

Moving a data record without a copy operation is functionally similar to the snapshot copy operation described above. A significant difference is that the virtual track directory entry **300** contains a NULL pointer in the virtual track address **320** to indicate that this virtual address does not contain any data and the source flag bit **306** is set to indicate that this virtual address is still a source. The following pseudo code listing indicates an instant move operation for a target data record, to highlight the difference between this operation and the above noted data record copy operation:

```
Read VTD Entry for New Target
Increase Free Space for Cylinder pointed to by
 old VTD Entry
Reorder Free List if necessary
Read VTD Entry for the Source
Set Source Flag in VTD Entry
Write a NULL pointer into the Logical Address
 Pointer of the VTD Entry
Write Updated VTD for Source back to VTD
Write an unmodified copy of the old VTD Entry
 to Target location in VTD
(This entry already has the Target Flag set)
Scan Source List for the Logical Address in the
 VTD Entry for the Target
Scan Target List to find this Target in Target
 List
Update the Target Entry to the address the data
 was moved to
 Move the Update Count Fields Flag from the
  command to the Target Entry
Journal the changes to the VTD and to the Copy
 Table
```

Virtual Track Directory Copy Implementation

This second method of managing the data pointers makes use of an expanded virtual track directory **3** which increases each entry **300** to allow room for a virtual track address **320** that consists of copy virtual device number **308**, copy virtual cylinder number **309** and copy virtual head number **310** elements which act as a pointer to another virtual track that was copied from the first virtual track. The virtual track directory entry for the track pointed to from the first virtual track directory entry contains the same logical address as the first and contains the virtual track address of the next virtual track directory entry in the chain of target data records. Thus, multiple tracks copied from a single source track are identified by a singly linked list that loops back to itself at the source track to form a synonym ring of pointers. Thus, the virtual track directory itself contains an embedded copy table instead of using the lookaside copy table described above. Theoretically, any number of copies of a single track can be

made using this method since the virtual track directory entries are simply linked together in ring form. As a management construct, the number of copies can be limited to a predetermined number and, if a user requests further copies to be made, a second set of copies can be created by staging the data record from the backend data storage devices to make a second physical copy in cache memory **113** which can be used as the basis of a second ring in order to enable the length of each ring to be maintained at a reasonable manageable number.

The operation of the virtual track directory implementation is illustrated in flow diagram form in FIG. **17**. At step **1001**, the data storage subsystem receives a copy request from data processor **2** over data channel **21**. Processor **204-0** in data storage subsystem **100** verifies at step **1002** that the extents are defined, the same length and do not overlap. The cache management software ensures at step **1003** that all the tracks in this target extent are cleared and available for the copy operation. This is explained in further detail in the following pseudo code:

```
For each track in source extent, search cache
  IF track is found
    Mark track as 'Copy Loop Track'
    IF track is modified
      CALL Copy Modified Track service routine
      PASS Source Virtual Track Address
      PASS Target Virtual Track Address
       Function forms Copy Loop in VTD and
       marks target as 'No Backend Address'
       IF Copy Loop is below Max Size
          RETURN (SUCCESS) ;* No Action
           Necessary
       ELSE (Loop too big - Need to Break
        Mark Target as Pseudosource in VTD
         Entry
        RETURN (Cache Copy to Target
         Address and Destage Target)
       ENDIF
       RECEIVE Status
       Cache must do the following:
       IF status is Cache Copy to Target and
        Destage Target
         Do Not search for track - target
          can't be in cache
         Do a Cache to Cache Copy of the
          source
         Load the copy with the target
          address
         Schedule the Destage of the track
       ENDIF
      ENDIF
    ENDIF
  ENDFOR
```

Once this operation is completed, the source and target virtual track directory entries are updated at step **1004** to indicate their status as source and target, respectively and the virtual track address information contained therein is modified at step **1005** to indicate that both of these virtual track directory entries are part of a copy loop.

In order to limit the length of the singly linked list of source and target tracks in the copy operation, the length of the copy list is checked at step **1006** and if less than a predetermined limit, the task is completed. If the copy list exceeds this predetermined limit, then at step **1007** a second copy loop is created as described in the following copy count management code:

```
IF the loop is bigger than limit
  Set 'Hold Off VCKD Response' flag
  Increment Copy Notify Count in Copy Command in
   Virtual Device Table
  IF any target is marked as Modified or as a
   pseudosource
    Mark Pseudosource as 'Notify when Destaged'
      Destage Task will tell Copy Task when the
      destage is complete and the Loop Size is
       reduced
    CALL Destage Track Cache function
      PASS Virtual Track Address
      PASS No Response Indicator
  ELSE (No tracks are modified)
    Mark Target as Pseudosource in VTD Entry
     (Set Source and Target)
    Mark Target (Pseudosource) as 'Notify when
     Destaged'
    CALL Stage and Destage Track cache service
     routine
      PASS Target Address
      Cache SW must hash to the passed address.
      IF the track is in cache
       Schedule the Destage of the track
      ELSE (Track is not in cache)
       Schedule the Stage of the track
       Once track is in cache, immediately
        schedule the Destage of the track
      ENDIF
      When track is destaged, Destage Task
       breaks Copy Loop
      into two Copy Loops with pseudosource as
       new source,
      and returns response to Copy Task.
  ENDIF
ENDIF
```

Staging and Destaging of Copy Loop Tracks

When a track is to be updated in cache memory **113**, it must be determined whether this track is part of a copy loop. It is important to do this to ensure that the integrity of the multiple copies of this track are maintained and that only the appropriate copies of this track are modified according to the following procedure:

```
IF the track is not a Copy Loop
    RETURN (SUCCESS)
     No action required by cache SW
ELSEIF (the track is a Target)
  IF the track marked as a Pseudosource in VTD
    RETURN (Do Not Update - Track Being
     Scheduled for Destage)
  ELSE
    Mark track as 'Modified In Cache' in VTD
     Entry
    RETURN (SUCCESS)
     No action required by cache SW
  ENDIF
ELSE (the track is a Source)
  Scan Copy Loop to find an unmodified target
  IF unmodified target found
   Could be marked 'No Backend Address'
    Mark Target as Pseudosource in VTD Entry
     (Set Source and Target)
    RETURN (Cache Copy to Returned Address and
     Destage Returned Track)
      Cache SW must hash to the returned
      address.
      IF the track is in cache
        Schedule the Destage of the track
      ELSE (Track is not in cache)
        DO a Cache to Cache Copy of the source
        Load the copy with the returned
         address (Pseudosource address)
        Schedule the Destage of the track
      ENDIF
      Go ahead with modifications to the source
  ELSE (unmodified target not found)
```

-continued

```
    RETURN (Target List with an indicator to
     Destage Targets)
        Cache SW must hash to the passed
        addresses and
        schedule the Destage of all those tracks.
        Destage will not allow the source track
        to be
        destaged until all the targets are
        destaged first.
        The Cache SW can go ahead with
        modifications to the source
  ENDIF
ENDIF
```

As can be seen from this pseudo code, a cache to cache copy of the track must be made if this track is a source in a copy loop in order to ensure that the noted copies of this track are maintained at their present status and not corrupted by the modification to the original track performed by cache memory **113**. Similarly, the destaging of copy loop tracks are performed in a manner to maintain the integrity of the copy loop and ensure that the proper vintage of data is written to the appropriate physical location in the backend data storage for the designated virtual address.

FIG. **18** illustrates in flow diagram form the operational steps taken by processor **204-0** when at step **1101** it schedules the writing of a virtual track that is a target to the backend storage **103**. At step **1102**, a check is made of the copy list to determine whether the target is the only target for the associated source. If not, at step **1103**, the target is destaged from cache memory **113** to backend storage **103** and the copy virtual track address **320** of the previous track in the copy list is updated to reflect the deletion of this target from the copy list. The target flag for the written target is reset at step **1107** to reflect the deletion of this target from the copy list. If this is the last target, at step **1104** it is destaged from cache memory **113** to backend storage **103**. At step **1106**, the source copy virtual track address **320** is deleted and the source and target flags are reset at step **1108** in the corresponding virtual track directory entries. The destaging algorithm for the copy list is described using the following pseudo code:

```
IF track is a source
    There must be at least one target for track
   to be a source
  IF all targets marked as 'No Backend Address'
    Write track to DASD
    Put Logical Address in Source and Targets in
     VTD
    Remove 'No Backend Address' indication
  ELSEIF (any Targets marked as 'Modified in
   Cache'
        AND Not marked 'Scheduled for Destage')
      Create TCB containing:
       Destage Failure Indicator
       Targets Not Destaged First Indicator
       Addresses of targets modified in cache and
        not scheduled for destage
      CALL Cleanup Track cache service routine
      PASS TCB Pointer
      The Targets may in fact have been
      scheduled for destage
      following the scheduling of the source.
      Cache SW must do following:
      FOR (All target addresses returned to
      cache)
        IF track not scheduled for destage
          CALL Destage Track Request
            PASS Returned Target Address
        ENDIF
        ENDFOR
        CALL Destage Track Request
          PASS Source Address
    ELSE (any Targets marked as 'Modified in Cache'
      AND marked 'Scheduled for Destage')
    Put Request on Destage Blocked Queue marked
     as
      'Do Not Destage until Modified Target
      Destaged'
    ENDIF
  ELSEIF (track is a pseudosource)
    Write track to DASD
    Update VTD to mark track as source
    Unlink previous source from Copy Loop
    Update Target Physical Addresses to new source
     location
    IF track marked as 'Notify when Destage'
      CALL Notify Copy Task function
        PASS Target Virtual Address
    ENDIF
  ELSEIF (track is a target)
    Must be modified
    Update VTD to mark track as 'Scheduled for
     Destage'
    Write track to DASD
    IF track marked as 'Notify when Destage'
      CALL Notify Copy Task function
        PASS Target Virtual Address
    ENDIF
    IF (this is the last modified target in the
     Copy Loop
     AND source is in Destage Blocked Queue marked
      as
        'Do Not Destage until Modified Target
        Destaged')
      Move source request to Destage Request Queue
      Write source track to DASD
    ENDIF
  ENDIF
```

As can be seen from these routines, care must be taken to not intermingle various versions of the virtual track instances as the copy loop is created, expanded and contracted by the movement of data into and out of cache memory **113** and the appropriate backend storage. A corresponding destaging process is executed for a copy table implementation of the pointer management.

Data Read Operation

FIGS. **6** and **7** illustrate in flow diagram form the operational steps taken by processor **204** in control unit **101** of the data storage subsystem **100** to read data from a data redundancy group **122-1** to **122-n+m** in the disk drive subsets **103**. The disk drive array data storage subsystem **100** supports reads of any size. However, the logical layer only supports reads of virtual track instances. In order to perform a read operation, the virtual track instance that contains the data to be read is staged from the logical layer into the cache memory **113**. The data record is then transferred from the cache memory **113** and any clean up is performed to complete the read operation.

At step **601**, the control unit **101** prepares to read a record from a virtual track. At step **602**, the control unit **101** branches to the cache directory search subroutine to assure that the virtual track is located in the cache memory **113** since the virtual track may already have been staged into the cache memory **113** and stored therein in addition to having a copy stored on the plurality of disk drives (**122-1** to **122-n+m**) that constitute the redundancy group in which the virtual track is stored. At step **603**, the control unit **101** scans the hash table directory of the cache memory **113** to determine

whether the requested virtual track is located in the cache memory **113**. If it is, at step **604** control returns back to the main read operation routine and the cache staging subroutine that constitutes steps **605–616** is terminated.

Assume, for the purpose of this description, that the virtual track that has been requested is not located in the cache memory **113**. Processing proceeds to step **605** where the control unit **101** looks up the address of the virtual track in the virtual to logical map table. At step **620**, control unit **101** determines whether the requested virtual track resides on secondary media by reviewing the contents of the virtual track directory as described above. If the requested virtual track is not on secondary media, processing advances to step **606** as described below.

Retrieve Logical Cylinder From Secondary Media

If the requested virtual track is archived on secondary media, control unit **101** branches to step **621** where it reads the secondary media directory, located in cache memory **113** to obtain the pointer indicative of the physical location of the requested virtual track, for example on magnetic tape **10A**. At step **622**, control unit **101** obtains an unused logical cylinder in disk drive array **100** to store the logical cylinder containing the requested virtual track, that is to be retrieved from the secondary media. At step **623**, control unit **101** sets the Retrieving flag in the secondary media directory to indicate that the logical cylinder is in the process of being transferred from the secondary media. Control unit **101** reads the logical cylinder containing the requested virtual track from its location in the secondary media to the reserved logical cylinder. Once the requested logical track has been transferred to the reserved logical cylinder, at steps **624** and **625**, control unit updates the status of this logical cylinder in the secondary media directory and virtual track directory, respectively.

Logical Track Staging

The control unit **101** allocates space in cache memory **113** for the data and relocates the logical address to the cache directory. At step **606**, the logical map location is used to map the logical device to one or more physical devices in the redundancy group. At step **607**, the control unit **101** schedules one or more physical read operations to retrieve the virtual track instance from appropriate ones of identified physical devices **122-1** to **122-n+m**. At step **608**, the control unit **101** clears errors for these operations. At step **609**, a determination is made whether all the reads have been completed, since the requested virtual track instance may be stored on more than one of the N+M disk drives in a redundancy group. If all of the reads have not been completed, processing proceeds to step **614** where the control unit **101** waits for the next completion of a read operation by one of the N+M disk drives in the redundancy group. At step **615** the next reading disk drive has completed its operation and a determination is made whether there are any errors in the read operation that has just been completed. If there are errors, at step **616** the errors are marked and control proceeds back to the beginning of step **609** where a determination is made whether all the reads have been completed. If at this point all the reads have been completed and all portions of the virtual track instance have been retrieved from the redundancy group, then processing proceeds to step **610** where a determination is made whether there are any errors in the reads that have been completed. If errors are detected then at step **611** a determination is made whether the errors can be fixed. One error correction method is the use of a Reed-Solomon error detection/correction code to recreate the data that cannot be read directly. If the errors cannot be repaired then a flag is set to indicate to the control unit **101** that the virtual track instance can not be read accurately. If the errors can be fixed, then in step **612** the identified errors are corrected and processing proceeds to step **630** where a test of the Collected Flag in the Logical Cylinder Table (LCT) is made. If the Collected Flag is clear, steps **631** and **632** are skipped and processing proceeds to step **604**. If the Collected Flag is set, processing proceeds to step **631** where the Logical Cylinder Access Counter is incremented and the Last Access Time/data is loaded with the current time and date. Processing then returns back to the main routine at step **604** where a successful read of the virtual track instance from the redundancy group to the cache memory **113** has been completed.

At step **617**, control unit **101** transfers the requested data record from the staged virtual track instance in which it is presently stored. Once the records of interest from the staged virtual track have been transferred to the data processor **2** that requested this information, then at step **618** the control unit **101** cleans up the read operation by performing the administrative tasks necessary to place all of the apparatus required to stage the virtual track instance from the redundancy group to the cache memory **113** into an idle state and control returns at step **619** to service the next operation that is requested.

Data Write Operation

FIG. **13** illustrates in flow diagram form the operational steps taken by the disk drive array data storage subsystem **100** to perform a data write operation. The disk drive array data storage subsystem **100** supports writes of any size, but again, the logical layer only supports writes of virtual track instances. Therefore in order to perform a write operation, the virtual track that contains the data record to be rewritten is staged from the logical layer into the cache memory **113**. The modified data record is then transferred into the virtual track modified and this updated virtual track instance is then scheduled to be written from the cache memory **113** where the data record modification has taken place into the logical layer. Once the backend write operation is complete, the location of the obsolete instance of the virtual track is marked as free space. Any clean up of the write operation is then performed once this transfer and write is completed.

At step **701**, the control unit **101** performs the set up for a write operation and at step **702**, as with the read operation described above, the control unit **101** branches to the cache directory search subroutine to assure that the virtual track into which the data is to be transferred is located in the cache memory **113**. Since all of the data updating is performed in the cache memory **113**, the virtual track in which this data is to be written must be transferred from the redundancy group in which it is stored to the cache memory **113** if it is not already resident in the cache memory **113**. The transfer of the requested virtual track instance to the cache memory **113** is performed for a write operation as it is described above with respect to a data read operation and constitutes steps **603–616** illustrated in FIG. **6** above.

At step **703**, the control unit **101** transfers the modified record data received from host processor **11** into

the virtual track that has been retrieved from the redundancy group into the cache memory **113** to thereby merge this modified record data into the original virtual track instance that was retrieved from the redundancy group. Once this merge has been completed and the virtual track now is updated with the modified record data received from host processor **11**, the control unit **101** must schedule this updated virtual track instance to be written onto a redundancy group somewhere in the disk drive array data storage subsystem **100**.

This scheduling is accomplished by the subroutine that consists of steps **705–710**. At step **705**, the control unit **101** determines whether the virtual track instance as updated fits into an available open logical cylinder. If it does not fit into an available open logical cylinder, then at step **706** this presently open logical cylinder must be closed out and written to the physical layer and another logical cylinder selected from the most free logical device or redundancy group in the disk drive array data storage subsystem **100**. At step **707**, the selection of a free logical cylinder from the most free logical device takes place. This ensures that the data files received from data processor **2** are distributed across the plurality of redundancy groups in the disk drive array data storage subsystem **100** in an even manner to avoid overloading certain redundancy groups while underloading other redundancy groups. Once a free logical cylinder is available, either being the presently open logical cylinder or a newly selected logical cylinder, then at step **708**, the control unit **101** writes the updated virtual track instance into the logical cylinder and at step **709** the new location of the virtual track is placed in the virtual to logical map in order to render it available to the data processors **2** - **2'**. At step **710**, the control unit **101** marks the virtual track instance that is stored in the redundancy group as invalid in order to assure that the logical location at which this virtual track instance is stored is not accessed in response to another data processor **2'** attempting to read or write the same virtual track. Since the modified record data is to be written into this virtual track in the cache memory **113**, the copy of the virtual track that resides in the redundancy group is now inaccurate and must be removed from access by the data processors **2** - **2'**. At step **711**, control returns to the main routine, where at step **712** the control unit **101** cleans up the remaining administrative tasks to complete the write operation. At step **713**, the processor **204** updates the free space directory to reflect the additional free space in the logical cylinder that contained the previous track instance and return to an available state at **714** for further read or write operations from data processor **2**.

Free Space Collection

When data in cache memory **113** is modified, it cannot be written back to its previous location on a disk drive in disk drive subsets **103** since that would invalidate the redundancy information on that logical track for the redundancy group. Therefore, once a virtual track has been updated, that track must be written to a new location in the data storage subsystem **100** and the data in the previous location must be marked as free space. Therefore, in each redundancy group, the logical cylinders become riddled with "holes" of obsolete data in the form of virtual track instances that are marked as obsolete. In order to create completely empty logical cylinders for destaging, the valid data in partially valid cylinders must be read into cache memory **113** and rewritten into new previously emptied logical cylinders. This process is called free space collection. The free space collection function is accomplished by control unit **101**. Control unit **101** selects a logical cylinder that needs to be collected as a function of how much free space it contains. The free space determination is based on the free space directory as illustrated in FIG. **8**, which indicates the availability of unused memory in data storage subsystem **100**. The table illustrated in FIG. **8** is a listing of all of the logical devices contained in data storage subsystem **100** and the identification of each of the logical cylinders contained therein. The entries in this chart represent the number of free physical sectors in this particular logical cylinder. A write cursor is maintained in memory and this write cursor indicates the available open logical cylinder that control unit **101** will write to when data is destaged from cache **113** after modification by associated data processor **2** or as part of a free space collection process. In addition, a free space collection cursor is maintained which points to the present logical cylinder that is being cleared as part of a free space collection process. Therefore, control unit **101** can review the free space directory illustrated in FIG. **8** as a backend process to determine which logical cylinder on a logical device would most benefit from free space collection. Control unit **101** activates the free space collection process by reading all of the valid data from the selected logical cylinder into cache memory **113**. The logical cylinder is then listed as completely empty and linked into the Free Cylinder List since all of the virtual track instances therein are tagged as obsolete. Additional logical cylinders are collected for free space collection purposes or as data is received from an associated data processor **2** until a complete logical cylinder has been filled. Once a complete logical cylinder has been filled, a new previously emptied logical cylinder is chosen.

FIG. **10** illustrates in flow diagram form the operational steps taken by processor **204** to implement the free space collection process. When Free Space collection has to be done, the best logical cylinder to collect is the one with the most sectors already free. This leads to the notion of a list of all of the logical cylinders in data storage subsystem **100** ordered by the amount of Free Space each contains. Actually, a list is maintained for each logical device, since it is desirable to balance free space across logical devices to spread virtual actuator contention as evenly as possible over the logical actuators. The collection of lists is called the Free Space Directory; the list for each logical device is called the Free Space List for the logical device. Each free space entry represents a logical cylinder. Each free space directory entry (FIG. **14**) contains a forward and backward pointer to create a double linked list as well. Each logical device's Free Space Link List is terminated by head and a tail pointers.

Each logical cylinder contains in its last few sectors a directory of its contents, called its Logical Cylinder Directory (LCD). This directory contains an entry for each virtual track instance contained within the logical cylinder. The entry for a virtual track instance contains the identifier of the virtual track and the identifier of the relative sector within the logical cylinder in which the virtual track instance begins. From this directory, the serial number of the logical cylinder instance, and the Virtual Track Directory, the Free Space Collection Process can determine which virtual track instances are still current in the logical cylinder and therefore need to

be moved to make the logical cylinder available for writing new data.

The basic process is initiated at step **1000** when processor **204** opens a logical cylinder to receive data collected, then proceeds to step **1001** where processor **204** selects a Logical Cylinder (LC) for collection based on the number of free logical sectors as listed in the Free Space Directory table of FIG. **8**. At step **1002**, processor **204** reads the logical cylinder directory for the logical cylinder that was selected at step **1001**. Processor **204** then at step **1003** reads the logical address from the virtual track directory (VTD) entry for each virtual track address that is contained in the read logical cylinder directory. At step **1005**, processor **204** compares the logical address that was stored in the virtual track directory entry with the logical address that was stored in the logical cylinder directory. If these two addresses do not match, that indicates the track instance is not valid for this virtual address and at step **1017** processor **204** determines that this track should not be relocated and execution exits.

If, at step **1005**, processor **204** determines that the virtual address stored in the virtual track descriptor matches the virtual address stored in the logical cylinder directory, at step **1006** the virtual track instance is staged into predetermined location in cache memory **113**. Processor **204** destages the virtual track instance to the disk drive subset **103** that contains the logical cylinder used by this free space collection process at step **1008**. At step **1011**, processor **204** updates the virtual track directory entry and exits at step **1020**. At step **1020**, processor **204** updates the free space directory to indicate that the collected cylinder is now a free cylinder available for data storage purposes and the data previously contained therein has been collected to a designated logical cylinder and the appropriate mapping table entries have been updated.

Enhanced Free Space Collection

Enhanced Free Space Collection occurs when a cylinder is collected that has already been collected before, as indicated by the Collected Flag in the Logical Cylinder Table (LCT). When data is collected and written to a cylinder separate from the normal destaging cylinder, that data is Read-Only or Low Access relative to the rest of the data in the Logical Cylinder, since any data that is updated is written to new cylinders. Data that is collected a second time is Read-Only or Low Access relative to all the data in the subsystem so it is Archive data. When Free Space Collection collects a cylinder that has not been collected before, the basic Free Space Collection Algorithm, as described in the previous section, is used. When Free Space Collection collects a cylinder that has the Collected or the Archive Flag in the LCT set, the Enhanced Free Space Collection Algorithm is used. FIG. **11** illustrates in flow diagram form the operational steps taken by processor **204** in control unit **101** of the data storage subsystem **100** to perform Enhanced Free Space Collection. The differences between Basic and Enhanced Free Space Collection are minor, but they are important to the hierarchical algorithm since they differentiate data into Low Access and Regular Access Logical Cylinders. In step **1100**, we allocate two logical cylinders to receive the data collected during free space collection. One cylinder is used for Low Access Data and the other is used for Regular Access Data. Steps **1001** through **1006** are the same as the basic algorithm. At step **1107** there is a test to determine if the virtual track that has been read from the cylinder being collected is Low Access. The track is low access if the Virtual Track Access Counter from the VTD divided by the age of the logical cylinder is below a low access threshold. The age of the Logical Cylinder is calculated by subtracting the Creation Data/Time (in the LCD) from the Current Data/Time. If the virtual track is low access, the data is written, at step **1108** to the low access logical cylinder. If the virtual track is not low access, the data is written, at step **1109** to the regular access logical cylinder.

Migrate Logical Cylinder

Data that is stored in Low Access Cylinders can be migrated to secondary media, such as magnetic tape **10A** or bulk disk storage, such as optical media. This is accomplished automatically and dynamically in disk drive array **100** by control unit **101**. The data migration process illustrated in FIG. **15** is initiated at step **1501** either periodically by control unit **101** to migrate data to secondary media on a regular basis or on a demand driven basis, such as when the number of available logical cylinders falls below a predetermined threshold or the number of relative versions of a generation data group exceeds a predetermined threshold. In either case, control unit initiates the migration process at step **1501** and selects a logical cylinder at step **1502**, identified as a low access cylinder by calculating the access rate from the last three fields in the Free Space Directory Entry as illustrated in FIG. **14**. At step **1503**, control unit **101** writes the selected logical cylinder to secondary media **10A**.

In operation, the selected logical cylinder is read from the redundancy group on which it is stored to cache memory **113** as described above. Once staged to cache memory **113**, the selected logical cylinder is transferred to secondary media **10A** via tape drive control unit **10** and data channel **20** in well-known manner as described above. Once the data write process is completed, control unit **101** at steps **1504**, **1505** updates the status of the secondary media directory and virtual track directory, respectively to indicate the archived nature of the migrated logical cylinder. At step **1506**, the logical cylinder in disk drive array **100** that stored the migrated logical cylinder is marked as free in the free space directory. The migration process concludes at step **1507** if no further logical cylinders are available for migration. Otherwise, the process of FIG. **15** is repeated.

Database Example

This example is to illustrate an application of the apparatus of the present invention. This example makes use of a database that is used for payroll purposes and contains the following files:

File Name	Contents
Personnel.Salaries	Salary information from personnel system.
Accounting.Income.Tax	IRS information regarding a specific payroll.
Payroll.Data	Actual payroll data.
Payroll.Data.Index	Index to the payroll database.
Meta.Data	Data used by the host processor to access data sets on that volume.

As illustrated in FIG. **1**, the data sets are initially located on volumes **11-1**, **11-2**, **11-3** in a distributed manner in the functional volumes that are directly ad-

dressable by the data processor 2. All of these data sets are related in that they are necessary elements of the payroll function. When the user creates the monthly payroll, the user application program 3 makes use of this set of data sets and therefore, all of these data sets must be temporally synchronized. In addition, if a user needed to recreate an old payroll run, due to an error that was found in that payroll run, all of these data sets are again required and these data sets must be temporally coordinated as of the time the user application program 3 ran the payroll which contained the error. The set of data sets must be temporally coordinated for each payroll iteration and therefore archive copies of this set of data sets must be time coordinated as well as the present version of this set of data sets.

To accomplish this, the user defines a snapshot application data group (SADG) that is associated with the above set of data sets. Assume for the purpose of this example, that the user defined this snapshot application data group using the name "PAYROLL.SADG". The records defining the PAYROLL.SADG would be contained in a SADG database on host processor 2 in the file server system catalog *5b*. The following tables illustrate typical contents of such a snapshot application data group definition.

TABLE 1

Payroll.SADG Definition	
Consists of	Payroll.Data
	Payroll.Data.Index
	Personnel.Salaries
	Accounting.Income.Tax
Keep 12 generations	
No expiration	
etc.	

TABLE 2

Payroll.SADG.G0025V01 Definition
As of 6/3/92 at 09:30:45:53
Source Volumes = Snapshot Volumes
V0001 = SV9356, 6 Segments
V0002 = SV9874, 12 Segments
V0003 = SV6783, 4 Segments

As can be seen from the example of Table 1, the file server system catalog *5b* contains snapshot application data group definition and administration information. In particular, the definition information includes a list of all of the data sets in this set of data group that comprise the PAYROLL.SADG application data group. In addition, administrative information such as the number of versions or instances of the snapshot application data group that should be maintained in file server system 1 is also defined. Further information of an administrative nature can also be stored in this definition file, such as the expiration date beyond which copies of data sets should not be stored, identification of preferred media to store certain instances of a generation data group, etc. The information contained in Table 2 notes the specific mapping of each instance of a snapshot application data group to the snapshot volume and while other tables note the particular physical storage location in file server system 1 wherein the set of data sets is stored. The mapping includes a definition of the snapshot volumes which are outside the functional address space of data processor 2 as well as the formula required to map each of these volumes.

To better understand the use of these tables and the apparatus described above, the following description notes the creation of a snapshot copy of a particular snapshot application data group as illustrated in flow diagram form in FIG. 20. At step **2001**, the data processor 2 terminates all activity on the data sets that are contained within the selected snapshot application data group that is to be copied. The termination of all activity is necessary because a plurality of user programs 3 can concurrently access the data sets contained within this set of data sets. Therefore, to ensure that a single temporally coordinated copy of the set of data sets is made, the data processor 2 terminates all access to the source data sets within this snapshot application data group as defined in the definitional information noted in Table 1. At step **2002**, if any of these data sets in this snapshot application data group are under the control of a database management system, the user program 3 in conjunction with data processor 2 uses the database management system utilities to check point these data sets to the database management system log files to ensure consistency of the snapshot application data group with respect to the database management system that operates on the selected data sets within this snapshot application data group. Processing advances to step **2003** where the user program 3 issues a snapshot copy command which is transmitted to the file server system utility 6. At step **2004**, the file server system utility 6 accesses the snapshot application data group definition records that are stored in catalog *5b* and as illustrated in Table 1 above to determine the processing options and the administrative handling required for this snapshot application data group. The file server system utility 6 also determines whether the user program 3 has access authorization to this snapshot application data group. At step **2005**, the file server system utility 6 determines the identity of the functional volumes that contain the source data sets which, in this example are functional volumes **11-1**, **11-2**, **11-3** which volumes are identified by the source volume indicia V0001, V0002, V0003, respectively. At step **2006**, the file server utility 6 issues a command to file server system 1 to create a snapshot volume for the identified source volume. This is accomplished by transmitting the command through data channel interface 7 over data channel 8 to file server system 1. At step **2007**, file server system 1 receives the issued command and accesses the mapping table stored therein to determine the availability of snapshot volume resources to execute the received command. If insufficient resources are available, the file server system 1 returns an error code at step **2013** to the file server system utility 6 which then aborts the snapshot copy operation. If there are sufficient resources available to make a snapshot volume copy of the identified source volume, at step **2008**, file server system 1 makes a copy of the pointers associated with the identified source volume as described above. The file server system 1 also returns a snapshot volume identification (SVID) and a number of segments of the snapshot volume that was consumed by the required copy as a measure of resource consumption to the file server system utility 6. At step **2009**, the file server system utility 6 updates the catalog information as illustrated in Table 2 to note the correspondence between the source volumes and the snapshot volumes for this particular instance of the snapshot application data group and also notes the number of segments consumed for each of the snapshot volumes. At step **2010**, file server system utility 6 determines whether additional source volumes need to be copied. If additional source

volumes need to be copies, processing returns to step **2006** and steps **2006–2009** are repeated until file server system utility **6** determines that no further source volumes remain to be copied. At this point, processing advances to step **2011** wherein the file server system utility **6** posts all records and notifies the user application program **3** that a new instance of this snapshot application data group has been created. At step **2012**, user application program **3** resumes normal processing of the source data sets. In this manner, the file server system **1** creates copies of the source volumes independent of the data processor **2** and these copies can be maintained on whatever media is designated by file server system **1** as described above. Therefore, the creation of instances of a snapshot application data group take place independent of the data processor **2** thereby reducing the load on the data processor **2** and enabling the file server system **1** to maintain the various instances of each snapshot application data group on various types of media without requiring changes to data processor **2** or the operating system **4** resident thereon. This provides the user with the capability of adding new media to file server system **1** or modifying the operation of file server system **1** without impacting the data processor **2** since the operation of file server system **1** is completely transparent to data processor **2**.

User Access to Snapshot Application Data Group

Once various instances of snapshot application data groups have been created by file server system **1**, the user can access these instances for a number of reasons in a very controlled manner. The first example is the automatic creation of tape backups of source data sets. This can be done as described above with regard to the archive memory **10** or, alternatively, can be done via data processor **2** in the traditional manner wherein the data processor **2** retrieves the data sets to be archived and transmits them via a data channel to an archive memory which is directly connected to data processor **2**. This would be accomplished by mounting the designated snapshot volumes as a functional volume directly addressable by data processor **2** using the file server system utilities. Once these snapshot volumes are mounted, data processor **2** can retrieve the data contained therein and transmit the retrieved data to the archive memory that is directly connected to data processor **2**.

A second application of the copies of a snapshot application data group is in the recovery of an application job failure wherein the data can be corrupted during processing. It is a common practice in batch processing jobs to make a copy of the critical data sets prior to update processing in the data processor **2** so that if the processing fails the copy of the data sets can be restored and used to replace the corrupted data on data processor **2**. In this application, the user application program **3** transmits commands to the file server system utility immediately prior to transmission of the data to the batch processing program on data processing **2**, to create a snapshot copy of the data that will be used in the batch processing. The snapshot copy operation is accomplished as described above and if there is a failure in the batch processing, the customer application program **3** can transmit commands to the file server system utility to request that the snapshot volumes be mounted in place of the source volumes to effectively delete all data on the source volumes and replace them with the data from the snapshot volumes. The meta data is also restored in this process and the customer has some administrative issues to address with regard to maintaining the meta data from the snapshot application data group instance that contained the corrupted data.

A third application of copies of a snapshot application data group is for the movement of data sets from one volume in file server system **1** to another volume that is external to file server system **1** or even to another volume within file server system **1** where file server system **1** is a hierarchical data storage system. The movement of data sets from one volume to another is typically for performance reasons to place the data sets on an appropriate media that corresponds to the needs of data processor **2** and the user application programs **3**. The transmission of the data sets from one volume to another can be accomplished through data processor **2** or can be done internally in file server system **1**, depending on the location of the target volume.

While a specific embodiment of this invention has been disclosed herein, it is expected that those skilled in the art can design other embodiments that differ from this particular embodiment but fall within the scope of the appended claims.

We claim:

1. A file server system for storing data sets for at least one data processor comprising:

a plurality of data storage volumes, each of which is capable of storing at least one data set received from a data processor;

means for maintaining data set pointers indicative of a set of data sets managed as a single data entity consisting of a plurality of interrelated ones of said data sets stored in first available memory space in a plurality of said data storage volumes in said file server system;

means, responsive to the subsequent receipt of a data set access request from a data processor identifying one of said data sets stored in said set of data sets, for creating a new version of said set of data sets that contains said requested data set, independent of said data set requesting data processor, including:

means for identifying a physical memory location of each data set in said set of data sets that contains said requested data set as specified by its data set pointer,

means for generating a new data set pointer, duplicative of said data set pointer, as the data set pointer for said copy of each said data set in said set of data sets,

means for maintaining data indicative of a correspondence between said data set pointers and said duplicative data set pointers, and

means for providing said data set requesting data processor with access to said set of data sets via said duplicative data set pointers.

2. The file server system of claim **1** wherein said creating means further comprises:

means, responsive to said data set requesting data processor modifying data in said requested data set, for copying said requested data set to second available memory space in another of said plurality of data storage volumes;

means for modifying said duplicative data set pointer to identify said second available memory space as a copy of said requested data set; and

means for updating said data set pointer to indicate that said data set, stored in said physical memory location is a prior version of said requested data set.

3. The file server system of claim 2 wherein said copying means copies a one of said plurality of said data storage volumes containing said requested data set to second available memory space in another of said plurality of data storage volumes.

4. The file server system of claim 1 wherein said creating means is further responsive to a subsequent receipt of another data set access request from a data processor identifying one of said data sets stored in said set of data sets and said duplicative set of data sets, for creating a third version of said requested data set, independent of said subsequent data set access requesting data processor.

5. The file server system of claim 1 further comprising:

means for storing threshold data indicative of the number of prior versions of each set of data sets that can be stored in said file server system.

6. The file server system of claim 5 further comprising:

means for creating a series of said data set pointers, said series being indicative of a time ordered sequence of prior versions of a particular set of data sets.

7. The file server system of claim 6 further comprising:

means, responsive to a generation of a next duplicative data set pointer for an identified set of data sets, for inserting said next duplicative data set pointer into a one of said series of data set pointers corresponding to said series associated with said identified set of data sets.

8. The file server system of claim 7 further comprising:

means, responsive to said file server system writing a set of data sets identified by one of said duplicative data set pointers in a selected available memory space, for deleting an oldest data set pointer from said series of set of data sets pointers when the number of pointers in said series exceeds said threshold.

9. The file server system of claim 8 further comprising:

cache memory means connected to and interconnecting said data processor and said data storage volumes for storing data sets transmitted therebetween;

archive memory means connected to said cache memory means for storing data sets that were previously stored in said file server system by said data processor;

means, responsive to the writing of a set of data sets in a selected available memory space, for comparing the number of set of data sets pointers in said series of set of data sets pointers, corresponding to said written set of data sets, to said threshold;

means, responsive to said number of set of data sets pointers exceeding said threshold, for signifying an oldest set of data sets pointer in said series of set of data sets pointers as archivable;

means for transferring said archivable set of data sets from said data storage volumes to said cache memory means; and

means for rewriting said cached archivable set of data sets into said archive memory means.

10. The file server system of claim 9 further comprising:

means, responsive to said file server system archiving a set of data sets identified by one of said set of data sets pointers in archive memory means, for deleting the set of data sets pointer corresponding to said archived set of data sets from said series of set of data sets pointers.

11. The file server system of claim 10 further comprising:

means, responsive to said rewriting means, for storing data indicative of the memory location in said archive memory means in which said archived set of data sets is stored.

12. The file server system of claim 10 further comprising:

means for determining the amount of said available memory space in said data storage volumes; and

means for resetting said threshold as a function of said determined available memory space.

13. The file server system of claim 1 further comprising:

means for storing data received from a user of said file server system indicative of the identity of each of said data sets in said set of data sets.

14. The file server system of claim 1 wherein said maintaining means operates independent of said at least one data processor.

15. The file server system of claim 1 wherein said maintaining means comprises:

means for configuring said plurality of data storage devices into a plurality of virtual data storage volumes, each of said virtual data storage volumes being capable of storing at least one data set thereon; and

means for presenting a data storage image of a selected one of said plurality of virtual data storage volumes to each of said at least one data processor.

16. The file server system of claim 15 wherein said presenting means transforms the format of said set of data sets containing said requested data set prior to enabling access to said requested data set by said data set requesting data processor.

17. The file server system of claim 15 wherein said creating means further comprises:

means for transferring said set of data sets containing said requested data set from a first data storage volume to a second data storage volume, wherein said first and second data storage volumes have different physical data storage characteristics.

18. The file server system of claim 1 wherein said maintaining means comprises:

means for configuring said plurality of data storage devices into m virtual data storage volumes, each of said virtual data storage volumes being capable of storing at least one data set thereon, wherein m is a positive integer greater than one; and

means for presenting a data storage image of n data storage volumes directly addressable by said at least one data processor, to said at least one data processor, wherein n is a positive integer greater than zero and less than m.

19. The file server system of claim 18 wherein said maintaining means further comprises:

means, responsive to a one of said at least one data processor requesting access to a selected data set in a set of data sets that is stored on a one of said m data storage volumes that is not directly addressable by said at least one data processor, for mounting said set of data sets containing said requested

data set on a one of said n directly addressable data storage volumes.

20. The file server system of claim 19 wherein said maintaining means further comprises:

means for transferring said set of data sets containing said requested data set from a first data storage volume having a first set of physical data storage characteristics to a second data storage volume having a second set of physical data storage characteristics.

21. The file server system of claim 1 wherein said plurality of data storage volumes comprises:

a plurality of disk drives for storing data thereon, a number of said disk drives being configured into at least two redundancy groups, each said redundancy group including n+m of said plurality of disk drives, where n and m are both positive integers with n greater than 1 and m at least equal to 1;

means for storing each of a plurality of data sets received from said data processor on successive ones of said n disk drives in a selected redundancy group;

means, responsive to said storing means storing data sets on all n disk drives in said selected redundancy group, for generating m segments of data redundancy information for said data sets stored on said n disk drives;

means for writing said m segments of redundancy data on to said m disk drives of said selected redundancy group; and

means, responsive to said writing means, for generating a data set pointer for each of said data sets stored on said n disk drives identifying the physical memory location of each said data set in said redundancy group.

22. In a file server system having a plurality of data storage volumes, each of which is capable of storing at least one data set received from a data processor, a method for storing data sets for at least one data processor, comprising the steps of:

maintaining data set pointers indicative of a set of data sets managed as a single data entity consisting of a plurality of interrelated ones of said data sets stored in first available memory space in a plurality of said data storage volumes in said file server system;

creating, in response to the subsequent receipt of a data set access request from a data processor identifying one of said data sets stored in said set of data sets, a new version of said set of data sets that contains said requested data set, independent of said data set requesting data processor, including:

identifying a physical memory location of each data set in said set of data sets that contains said requested data set as specified by its data set pointer,

generating a new data set pointer, duplicative of said data set pointer, as the data set pointer for said copy of each said data set in said set of data sets,

maintaining data indicative of a correspondence between said data set pointers and said duplicative data set pointers, and

providing said data set requesting data processor with access to said set of data sets via said duplicative data set pointers.

23. The method of claim 22 wherein said step of creating further comprises:

copying, in response to said data set requesting data processor modifying data in said requested data set, said requested data set to second available memory space in another of said plurality of data storage volumes;

modifying said duplicative data set pointer to identify said second available memory space as a copy of said requested data set; and

updating said data set pointer to indicate that said data set, stored in said physical memory location is a prior version of said requested data set.

24. The method of claim 23 wherein said step of copying copies a one of said plurality of said data storage volumes containing said requested data set to second available memory space in another of said plurality of data storage volumes.

25. The method of claim 22 wherein said step of creating is further responsive to a subsequent receipt of another data set access request from a data processor identifying one of said data sets stored in said set of data sets and said duplicative set of data sets, for creating a third version of said requested data set, independent of said subsequent data set access requesting data processor.

26. The method of claim 22 further comprising the step of:

storing threshold data indicative of the number of prior versions of each set of data sets that can be stored in said file server system.

27. The method of claim 26 further comprising the step of:

creating a series of said data set pointers, said series being indicative of a time ordered sequence of prior versions of a particular set of data sets.

28. The method of claim 27 further comprising the step of:

inserting, in response to a generation of a next duplicative data set pointer for an identified set of data sets, said next duplicative data set pointer into a one of said series of data set pointers corresponding to said series associated with said identified set of data sets.

29. The method of claim 28 further comprising the step of:

deleting, in response to said file server system writing a set of data sets identified by one of said duplicative data set pointers in a selected available memory space, an oldest data set pointer from said series of set of data sets pointers when the number of pointers in said series exceeds said threshold.

30. The method of claim 29, wherein said file server system includes a cache memory connected to and interconnecting said data processor and said data storage volumes for storing data sets transmitted therebetween and an archive memory connected to said cache memory for storing data sets that were previously stored in said file server system by said data processor, the method further comprising the steps of:

comparing, in response to the writing of a set of data sets in a selected available memory space, the number of set of data sets pointers in said series of set of data sets pointers, corresponding to said written set of data sets, to said threshold;

signifying, in response to said number of set of data sets pointers exceeding said threshold, an oldest set of data sets pointer in said series of set of data sets pointers as archivable;

transferring said archivable set of data sets from said data storage volumes to said cache memory; and

rewriting said cached archivable set of data sets into said archive memory.

31. The method of claim 30 further comprising the step of:

deleting, in response to said file server system archiving a set of data sets identified by one of said set of data sets pointers in archive memory, the set of data sets pointer corresponding to said archived set of data sets from said series of set of data sets pointers.

32. The method of claim 31 further comprising the step of:

storing, in response to said step of rewriting, data indicative of the memory location in said archive memory in which said archived set of data sets is stored.

33. The method of claim 31 further comprising the steps of:

determining the amount of said available memory space in said data storage volumes; and

resetting said threshold as a function of said determined available memory space.

34. The method of claim 22 further comprising the step of:

storing data received from a user of said file server system indicative of the identity of each of said data sets in said set of data sets.

35. The method of claim 22 wherein said step of maintaining operates independent of said at least one data processor.

36. The method of claim 22 wherein said step of maintaining comprises:

configuring said plurality of data storage devices into a plurality of virtual data storage volumes, each of said virtual data storage volumes being capable of storing at least one data set thereon; and

presenting a data storage image of a selected one of said plurality of virtual data storage volumes to each of said at least one data processor.

37. The method of claim 36 wherein said step of presenting transforms the format of said set of data sets containing said requested data set prior to enabling access to said requested data set by said data set requesting data processor.

38. The method of claim 36 wherein said step of creating further comprises:

transferring said set of data sets containing said requested data set from a first data storage volume to a second data storage volume, wherein said first and second data storage volumes have different physical data storage characteristics.

39. The method of claim 22 wherein said step of maintaining comprises:

configuring said plurality of data storage devices into m virtual data storage volumes, each of said virtual data storage volumes being capable of storing at least one data set thereon, wherein m is a positive integer greater than one; and

presenting a data storage image of n data storage volumes directly addressable by said at least one data processor, to said at least one data processor, wherein n is a positive integer greater than zero and less than m.

40. The method of claim 39 wherein said step of maintaining further comprises:

mounting, in response to a one of said at least one data processor requesting access to a selected data set in a set of data sets that is stored on a one of said m data storage volumes that is not directly addressable by said at least one data processor, said set of data sets containing said requested data set on a one of said n directly addressable data storage volumes.

41. The method of claim 40 wherein said step of maintaining further comprises:

transferring said set of data sets containing said requested data set from a first data storage volume having a first set of physical data storage characteristics to a second data storage volume having a second set of physical data storage characteristics.

42. The method of claim 22 wherein said plurality of data storage volumes comprises a plurality of disk drives for storing data thereon, a number of said disk drives being configured into at least two redundancy groups, each said redundancy group including n+m of said plurality of disk drives, where n and m are both positive integers with n greater than 1 and m at least equal to 1, said method further comprises the steps of:

storing each of a plurality of data sets received from said data processor on successive ones of said n disk drives in a selected redundancy group;

generating, in response to said step of storing data sets on all n disk drives in said selected redundancy group, m segments of data redundancy information for said data sets stored on said n disk drives;

writing said m segments of redundancy data on to said m disk drives of said selected redundancy group; and

generating, in response to said step of writing, a data set pointer for each of said data sets stored on said n disk drives identifying the physical memory location of each said data set in said redundancy group.

* * * * *